EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103004/B/2021/421000214325

PARSOVANIE ALGORITMOV DO VYBRANÉHO PROGRAMOVACIEHO JAZYKA

Bakalárska práca

2021 **Jozef Šulek**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

PARSOVANIE ALGORITMOV DO VYBRANÉHO PROGRAMOVACIEHO JAZYKA

Bakalárska práca

Študijný program:	Hospodárska informatika
Študijný odbor:	**6292** Hospodárska informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Vedúci bakalárskej práce:	RNDr. Eva Rakovská, PhD.

Bratislava, 2021 **Jozef Šulek**

Ekonomická univerzita v Bratislave
Fakulta hospodárskej informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Jozef Šulek
Študijný program: hospodárska informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: Bakalárska záverečná práca
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Parsovanie algoritmov do vybraného programovacieho jazyka

Anotácia: Softvér PS diagram je aplikácia na výučbu algoritmizácie úloh a bol navrhnutý tak, že študentovi umožňuje vytvoriť vývojový diagram a parsuje ho následne do Pascalu. Aplikáciu je možné upraviť a pridať parsovanie do iného programovacieho jazyka. Práca sa zaoberá návrhom parsovania algoritmov zo softvéru PS Diagram do vybraného programovacieho jazyka.

Vedúci: RNDr. Eva Rakovská, PhD.
Katedra: KAI FHI - Katedra aplikovanej informatiky FHI
Vedúci katedry: Ing. Mgr. Peter Schmidt, PhD.

Dátum zadania: 24.03.2020

Dátum schválenia: 24.03.2020 Ing. Mgr. Peter Schmidt, PhD.
vedúci katedry

ČESTNÉ PREHLÁSENIE

Čestne vyhlasujem, že záverečnú prácu som vypracoval samostatne a že som uviedol všetku použitú literatúru.

Dátum: .....

Jozef Šulek

POĎAKOVANIE

Touto cestou si dovoľujem poďakovať sa vedúcej bakalárskej práce RNDr. Eve Rakovskej, PhD., za ľudský prístup, odbornú pomoc, pripomienky a cenné rady, ktoré mi poskytovala pri vypracovaní záverečnej bakalárskej práce.

ABSTRAKT

Šulek, Jozef : Parsovanie algoritmov do vybraného programovacieho jazyka. – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra aplikovanej informatiky. – Konzultant Eva Rakovská, RNDr. PhD. – Bratislava: FHI EU, rok 2021, 50 strán.

Cieľom tejto bakalárskej práce je názorne ukázať konverziu algoritmov v podobe vývojových diagramov do konkrétnej implementácie v zvoliteľnom programovacom jazyku, a to konkrétne rozšírením programu PS Diagram.

Program PS Diagram sa využíva na akademické účely a cieľom tejto práce je tento program rozšíriť o možnosť prekladania do programovacích jazykov, ktoré sú v súčasnej dobe najpoužívanejšie: Java a Python.

Cieľom práce je taktiež prispieť do programu s otvorenou licenciou PS Diagram, ktorý, po možnom rozšírení, sa môže stať softvérovým nástrojom pre zrýchlenie a skvalitnenie návrhu a implementácie softvéru, či už pre akademické alebo komerčné účely.

Práca je rozdelená do 3 kapitol. Obsahuje 11 obrázkov a 10 príloh. Pri Implementácií tejto záverečnej práce bolo napísaných 6303 riadkov zdrojového kódu v Jave.

Prvá kapitola je venovaná algoritmom a dátovým štruktúram. Nasledujúca kapitola je venovaná PS Diagramu a jeho významu pre akademické účely. V ďalšej kapitole sa venujeme rozhraniu ANTLR (ANother Tool for Language Recognition) a jeho použitiu pri rozpoznávaní písma a gramatiky. Záverečná kapitola prezentuje a opisuje konkrétne rozšírenie PS Diagramu, ktoré sme navrhli a implementovali.

Výsledkom riešenia danej problematiky je PS Diagram rozšírení o možnosť prekladať z a do programovacích jazykov Java a Python.

Kľúčové slová :

Algoritmy, Dátové štruktúry, Vývojové diagramy, Java, Python, PS Diagram, ANTLR

ABSTRACT

Šulek, Jozef: Parsing algorithms to selected programming language. – The University of Economics in Bratislava. Faculty of Economic Informatics; Department of Applied Informatics. - Tutor of bachelor thesis: RNDr. Eva Rakovská PhD.. – Bratislava: FHI EU, 2021, 50 pages

The aim of this thesis is presentation of the conversion of algorithms in the form of development diagrams into a particular implementation in the selected programming language, specifically by extending the PS Diagram program.

Program PS Diagram is used for academic purposes and the aim of this work is to extend this program for the possibility of translating into currently the most widely used programming languages, Java and Python.

The aim of the thesis is also to contribute to the open source program, PS Diagram, which, after possible extension, can become a software tool for acceleration and improvement of the design and implementation of software, whether for academic or commercial purposes.

The work is divided into 3 chapters. It contains 11 images and 10 additions. For implementation of this thesis, 6303 lines of source code in Java were written.

The first chapter is devoted to algorithms and data structures. The following chapter is devoted to the PS Diagram and its importance for academic purposes. In the next chapter we deal with the ANTLR (ANother Tool for Language Recognition) framework and its use in recognizing language and grammar. The final chapter presents and describes the specific extension of the PS Diagram that we designed and implemented.

The solution of the issue is PS Diagram extended for the possibility of translating from and into Java and Python programming languages.

Key words: Algorithms, Data Structures, Development Diagrams, Java, Python, PS Diagram, ANTLR

Obsah

Úvod

Hlavným cieľom tejto práce je rozšírenie PS Diagramu o možnosť prekladania vývojových diagramov do a z programovacích jazykov Java a Python. PS Diagram je aplikácia s otvorenou licenciou, používaná na vizualizáciu algoritmov. Pre dosiahnutie hlavného cieľa je taktiež potrebné analyzovať dané rozšírenie podľa užívateľských a softvérových požiadaviek. Po analýze nadväzuje tvorba návrhu podľa princípov softvérového inžinierstva a pomocou UML diagramov. Vytvorený návrh je potrebné implementovať.

V prvej časti tejto práce je prezentovaný koncept analýzy, návrhu a optimalizácie algoritmov a dátových štruktúr. Taktiež je naznačená dôležitosť danej disciplíny pre viaceré vedné aj podnikové oblasti.

Ďalšia časť patrí programu s otvorenou licenciou PS Diagram, a jeho využitiu pre vizualizáciu a prácu s algoritmami v podobe vývojových diagramov a zdrojového kódu rôznych programovacích jazykov.

Následne je predstavená technológia ANTLR, ktorá sa využíva ako prostriedok pre rozpoznávanie jazyka. Tento softvérový nástroj je využitý pri implementácií prekladania zdrojového kódu do vývojových diagramov.

V záverečnej kapitole je detailne popísaná konkrétna analýza, návrh a implementácia, ktoré bolo potrebné vykonať s cieľom rozšíriť program PS Diagram o možnosť prekladania vývojových diagramov do a z programovacích jazykov Java a Python.

1. Súčasný stav problematiky doma a v zahraničí

1.1 Algoritmy a dátové štruktúry

Algoritmus je presne stanovený postup, zostavený po sebe nasledujúcimi krokmi, pre riešenie daného problému. Algoritmus je návod na vykonanie činnosti, ktorý na základe vstupných údajov povedie v konečnom čase k výsledku. Algoritmus je elementárnym pojmom informatiky. Algoritmus má svoje vlastnosti, a to sú nasledovné:

- Elementárnosť – postup je zložený z jednoduchých krokov, ktoré sú pre vykonávateľa algoritmu zrozumiteľné.
- Determinovanosť - postup je zostavený tak, že v každom momente jeho vykonávania je jednoznačne určené, aká činnosť má nasledovať, alebo či sa postup skončil.
- Konečnosť – postup skončí po vykonaní konečného počtu krokov (v konečnom čase).
- Rezultatívnosť – postup vedie po vykonaní konečného počtu krokov ku vyriešeniu úlohy.
- Hromadnosť – algoritmus je použiteľný pre celú triedu prípustných problémov a vstupných údajov.
- Efektívnosť - výpočet sa uskutočňuje v čo najkratšom čase a s využitím čo najmenšieho množstva prostriedkov (časových i pamäťových). [2]

Algoritmy sú základnými stavebnými kameňmi programovania. Ruka v ruke s algoritmami idú dátové štruktúry. [1]

Dátové štruktúry sú schémy pre organizáciu údajov. Dátové štruktúry sú programovým spôsobom ukladania údajov, aby bolo možné údaje efektívne využívať. [2]

Dátová štruktúra je formát organizácie, spracovania a ukladania údajov, ktorý umožňuje efektívny prístup a úpravy. Presnejšie povedané, dátová štruktúra je súbor dátových hodnôt, vzťahov medzi nimi a funkcií alebo operácií, ktoré je možné na manipuláciu údajov použiť. [3]

Pre manipuláciu s dátovými štruktúrami sú dôležité nasledujúce kategórie algoritmov:

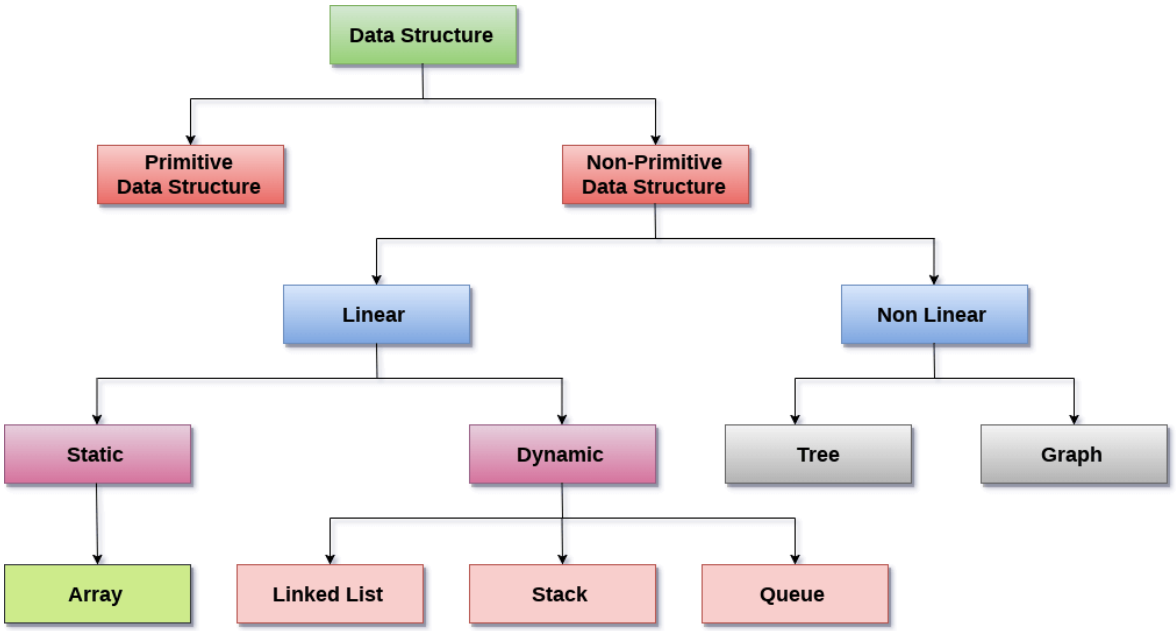
- Prehľadávanie – Algoritmy pre hľadanie prvku v dátovej štruktúre, napríklad exponenciálne alebo binárne prehľadávanie.
- Zoraďovanie – Algoritmy pre zoraďovanie prvkov v dátovej štruktúre, napríklad merge sort alebo quick sort algoritmy.
- Vkladanie – Algoritmy pre vkladanie (pridávanie) prvkov do dátovej štruktúry.
- Menenie – Algoritmy pre menenie prvkov v dátovej štruktúre.
- Mazanie – Algoritmy pre mazanie prvkov v dátovej štruktúre.[1]

Štúdium algoritmov a dátových štruktúr je základom každého učebného plánu informatiky, ale nie je to len pre programátorov a študentov počítačových vied. Algoritmy majú v dnešnej dobe široké uplatnenie vo viacerých vedných disciplínach ako aj v komerčnej sfére. V odvetviach fyziky a biológie sa základné metódy návrhu a optimalizácie algoritmov stali nevyhnutnými vo vedeckom výskume; od architektonických modelovacích systémov po simuláciu lietadiel sa stali základnými nástrojmi v strojárstve; od databázových systémov po internetové vyhľadávače sa stali podstatnou súčasťou moderných softvérových systémov. [1]

Algoritmy a dátové štruktúry naberajú na aktuálnosti a dôležitosti ako na akademickej pôde tak aj v podnikovej praxi, s prudko rastúcimi objemami a rýchlosťou údajov sú na dnešný softvér kladené vysoké požiadavky z hľadiska optimalizácie časovej a pamäťovej náročnosti. Aj napriek rastúcej výpočtovej sile procesorov a čoraz väčšej dostupnej operačnej pamäti, bez softvéru s efektívne navrhnutými algoritmami by čoraz lepší hardvér nestačil na riešenie problémov ohľadom Big Data alebo iných trendov v informačných technológiách.[4]

Vyššie programovacie jazyky ponúkajú základné dátové štruktúry a algoritmy v základných knižniciach jazyka. V Jave sú to napr. dátové štruktúry pole (Array). Zoznam (List), mapa (Map), množina (Set), rada (Queue), zásobník (Stack) a ich variácie aj mnohé iné. A algoritmy ako napr. zoradenie, zistenie prítomnosti prvku, filtrovanie, nájdenie minima alebo maxima, rôzne množinové operácie, operácie s reťazcami a podobne. Programátor však musí vedieť, ktoré z nich si vybrať a preto poznať ich vlastnosti a obmedzenia, prípadne aj časovú a pamäťovú náročnosť. Často sa taktiež v praxi môže stretnúť s problémami, ktoré vyžadujú vlastné dátové štruktúry a algoritmy. [4][5]

Pri optimalizácií programov záleží vo veľkej miere na správnom návrhu algoritmov a dátových štruktúr. Pre tieto účely nám môžu poslúžiť znalosti a vedomosti z danej oblasti a taktiež rôzne nástroje pre vizualizáciu a organizovanie algoritmov pomocou vývojových diagramov, ako napr. draw.io, lucidchart, microsoft word, pero a papier alebo PS Diagram.



Obrázok č.1 **Klasifikácia dátových štruktúr**

Zdroj:https://bcastudyguide.wordpress.com/unit-1-introduction-to-data-structure-and-its-characteristics/

Na obrázku č.1 môžeme vidieť základné rozdelenie bežne používaných dátových štruktúr. Prvé rozdelenie sú primitívne a neprimitívne dátové štruktúry, pričom primitívne môžeme nahradiť názvom dátové typy napr. int, float, byte, char, boolean a podobne. Neprimitívne sa ďalej delia na lineárne, a teda sekvenčne usporiadané a nelineárne, kde každý prvok je priamo spojený s viacerými prvkami dátovej štruktúry. Lineárne sa ďalej delia na statické a dynamické, statické nemenia dynamicky svoju veľkosť, takúto vlastnosť má napríklad pole. Dynamické môže dynamicky meniť svoju veľkosť počas behu programu resp. počas vykonávania algoritmu, sú to napríklad lineárny zoznam, zásobník alebo rad. Nelineárne dátové štruktúry sú napr. stromy a grafy. (samotné rozdelenie je príkladom stromovej dátovej štruktúry)

1.2 PS Diagram

PS Diagram je softvérová aplikácia pre rýchlu a jednoduchú tvorbu algoritmov v podobe vývojových diagramov, ako uvádza autor, Miroslav Bartyzal. Vývojové diagramy sú všeobecne používaný spôsob výučby pre návrh a analýzu algoritmov. Pokým neprišiel PS Diagram väčšina škôl na Slovensku a Česku používala ako pomôcku pre tvorbu vývojových diagramov papier a pero alebo Microsoft Word. Nedostatky jedného aj druhého z týchto nástrojov efektívne rieši aplikácia PS Diagram. PS Diagram bol od 27.06.2018 do 14.03.2021 stiahnutí 15004 krát, vyslúžil si viacero pozitívnych recenzií a stal sa súčasťou viacerých akademických publikácií a internetových tutoriálov týkajúcich sa algoritmov.[19][20] PS Diagram získal aj viacero ocenení, Diplomová práca roku 2012, Diplomky na stojáka 2013, a ŠVOČ Slovensko 2013. [6]

PS Diagram okrem tvorby vývojových diagramov ponúka viacero rôznych funkcií. Počas tvorby algoritmu ponúka aplikácia automatické formátovanie, upozornenia, odporúčania a chybové hlášky. Po vytvorení vývojového diagramu môžeme diagram zamknúť a sledovať jeho priebeh, a to buď po jednotlivých krokoch, alebo automaticky zaradom sledovať animáciu všetkých krokov. Počas animácie vidíme stav jednotlivých premenných a tak isto pri cykloch a vetveniach diagramu vidíme priebeh vývoja.

Za kľúčovú funkciu PS Diagramu môžeme pokladať aj import a export zdrojového kódu.

Pri importe zdrojového kódu si používateľ vyberie jazyk, v ktorom je implementovaný blok kódu, ktorý chce vložiť. Následne napíše alebo nakopíruje zdrojový kód do okienka aplikácie a PS Diagram vytvorí vývojový diagram reprezentujúci algoritmus implementovaný v zdrojovom kóde.

Pri exporte zdrojového kódu si používateľ vyberie jazyk a po stlačení tlačidla export sa do aplikačného okienka vypíše zdrojový kód reprezentujúci implementáciu algoritmu navrhnutú aktuálnym vývojovým diagramom v aplikácií. [6]

Pri analýze a návrhu tejto záverečnej práce existovala pri výbere programovacieho jazyka iba jedna možnosť a to Pascal, ktorý bol pred rokmi používaný pri výučbe programovania, dnes je však na ústupe.[12]

Podľa viacerých prieskumov patria jazyky Python a Java medzi najpoužívanejšie programovacie jazyky všeobecne, a takisto sú často používané pri výučbe programovania na stredných školách a univerzitách. [12][13]

Po úspešnej implementácií rozšírenia PS Diagramu, nadobudne PS Diagram novú možnosť jednoduchého prevodu kódu z jedného programovacieho jazyka do druhého, medzi jazykmi Pascal, Java a Python.

Taktiež bude možné algoritmy vizualizovať, meniť, optimalizovať v podobe vývojových diagramov namiesto podoby zdrojového kódu, čo môže byť v niektorých prípadoch značne prehľadnejšie. PS Diagram by tak mohol mať širšie uplatnenie v akademickej a podnikovej praxi.

Z vyššie uvedených dôvodov sme sa v tejto záverečnej práci rozhodli rozšíriť PS Diagram o možnosť importu a exportu do jazykov Java a Python.

1.3 Porovnanie programovacích jazykov

Programovacie jazyky môžu byť klasifikované podľa programovacích paradigiem, resp. štýlov vývoja softvéru, ktorý daný programovací jazyk podporuje, resp. je navrhnutý tak, aby bol programátorovi uľahčený proces tvorby softvéru danou programovacou paradigmou. Známe delenie je na funkcionálne alebo objektovo orientované programovacie jazyky. Exitujú taktiež procedurálne, imperatívne, deklaratívne, udalosťami riadené programovacie jazyky a iné. [15][18]

Programovacie jazyky väčšinou podporujú viaceré paradigmy, napr. v Jave je možné vytvárať aplikácie funkcionálnym alebo procedurálnym spôsobom, a nepoužívať pri vývoji objektovo orientovanú paradigmu, aj keď je Java všeobecne považovaná za objektovo orientovaný programovací jazyk. Python môže byť taktiež použitý na vytváranie skriptov, funkcionálne programovanie, ale pri vývoji je možné taktiež skrývať aplikačnú a dátovú logiku do tried a objektov. Pascal je procedurálny a imperatívny programovací jazyk.[5][14]

Programovacie jazyky môžu byť taktiež rozdelené na vyššie a nižšie jazyky. Toto rozdelenie odkazuje na úroveň abstrakcie od hardvéru a úmerne k tomu úrovni priblíženia sa ľudskej reči. Java, Python aj Pascal patria medzi vyššie programovacie jazyky, pretože

ich príkazy sú písané s vyššou úrovňou abstrakcie od strojového kódu. Príkladom nižšieho programovacieho jazyka je napr. Assembler, ktorého syntax je dosť podobná strojovému kódu.[16][5][14]

Inštrukcie vyšších programovacích jazykov musia byť preložené do príkazov, ktoré je počítač schopný vykonať, resp. do strojového kódu. Jazyky môžu byť kompilované alebo interpretované. Kompilované jazyky ako Go-lang alebo C++, sú preložené do natívneho strojového kódu kompilačným programom, ešte pred začatím ich spustenia. Interpretované jazyky sú prekladané postupne počas behu programu a nevyžadujú predchádzajúcu kompiláciu, interpreter operuje priamo na zdrojovom kóde. Kompilované jazyky sú pokladané za rýchlejšie pričom interpretované za kompatibilnejšie. Pascal je všeobecne považovaný za kompilovaný jazyk, aj keď je vytvorených aj niekoľko programov slúžiacich na jeho interpretáciu. Python môže byť skompilovaný a následne spúšťaný alebo môže byť aj interpretovaný. Java a JVM (Java Virtual Machine) má snahu o kombináciu benefitov obidvoch prístupov. Zdrojový kód napísaný v Jave je najskôr skompilovaný do Java Byte Code, špeciálneho strojového jazyka zrozumiteľného pre JVM, a pri spúšťaní programu je interpretovaný, v novších verziách Javy JIT(Just In Time) interpreterom, so snahou o optimalizáciu výkonu. Pre spôsob akým emulátor jazyka Java kompiluje a interpretuje zdrojový kód na strojový kód je pokladaná za kompatibilný a aj pomerne výkonný.[17][5]

Pri analýze a návrhu vlastných interpreterov jazykov Python a Java do jazyka PS Diagram sú využité vlastnosti daných jazykov popísané v tejto kapitole spolu so syntaktickými pravidlami.

2. Cieľ práce, metodika práce a metódy skúmania

2.1 Cieľ práce

Táto bakalárska práca má ako hlavný cieľ vývoj nových softvérových modulov pre program PS Diagram, ktoré slúžia na prekladanie vývojových diagramov do a z programovacích jazykov. Na dosiahnutie tohto cieľa je potrebné naplniť aj viaceré čiastkové ciele.

Jedným z čiastkových cieľov je charakterizovať algoritmy a dátové štruktúry ako také. Takisto je potrebné urobiť rešerš dostupných štandardov a technológií používaných pri návrhu a vizualizácií algoritmov a dátových štruktúr. Tieto čiastkové ciele sú postupne splnené v rámci druhej, tretej a štvrtej kapitoly.

Obmedzenia práce:

Pri realizácií tejto práce a hlavne pri implementácií rozšírenia programu PS Diagram sa v značnej miere používa rozpoznávanie jazyka. Rozpoznávanie jazyka je v informatike disciplína, ktorá je pokladaná za komplikovanú a náročnú a nie sú na ňu úplné, zaručené a hotové riešenia.[7] Softvérový nástroj ANTLR (ANother Tool for Language Recognition), ktorý je v PS Diagrame použitý, má taktiež svoje hranice a nedostatky. Jednotlivé znaky a slová majú v rôznom kontexte rôzny význam. Rôzne časti vstavaných gramatických pravidiel pre programovacie jazyky Java a Python sme upravili, lebo v pôvodnej verzií ANTLR neponúkali požadované správanie, pri prekladaní sa vyskytovali chyby.

Program PS Diagram je schopný zostrojiť jeden diagram, na ktorý je schopný použiť telo jednej metódy. Preto prekladanie väčších častí kódu nie je možné. Pri návrhu možných budúcich rozšírení v tejto práci navrhujeme možnosť viacerých inštancií, ktoré by si preposielali správy a údaje, a takým spôsobom by bolo možné spracovávať viacero metód, volajúcich sa navzájom.

Program PS Diagram nedisponuje rozpoznávaním a deklaráciou dátových typov pre interaktívne vývojové diagramy. V pôvodnej verzii programu, program pri miestach kde by mali byť dátové typy vypíše upozornenie. Pri jazyku Java program implicitne odvodzuje dátové typy premenných podľa hodnôt, ktoré sú im priradené. Pri importe a následnom exporte sa preto môžu vyskytnúť prípady zmeny dátového typu, avšak so

zachovaním funkčnosti, z numerického sa nestane textový dátový typ tam, kde by to spôsobilo chybu. Je to preto, že implicitné odvodzovanie sa uskutočňuje podľa príkazov, v ktorých je daná premenná použitá a hodnôt, ktoré nadobúda.

Modul import a export podporuje iba základnú syntax jazyka, vytvorenie poľa, cykly a podmienky. Avšak nepodporuje rôzne knižničné funkcie jazyka Python ani jazyka Java (rôzne Collections a Streams, niektoré Stringové knižničné funkcie), ktoré by mohli byť očakávané.

2.2 Metodika práce a metódy skúmania

Analýza

Pre realizáciu tejto práce bolo potrebné analyzovať program PS Diagram, jeho vnútornú štruktúru, požadované fungovanie, obmedzenia a proces tvorby a spájania objektov vývojových diagramov. Táto analýza je popísaná v poslednej časti tejto bakalárskej práce v časti softvérová analýza.

Bolo taktiež potrebné analyzovať softvérový nástroj pre rozpoznávanie jazyka ANTLR (ANother Tool for Language Recognition), základné fungovanie, gramatické pravidlá zo syntaktického a sémantického hľadiska a taktiež dostupné gramatické súbory pre programovací jazyk Java a Python. Táto analýza je popísaná v podkapitole Rozpoznávanie jazyka a ANTLR.

Ešte pred návrhom bolo potrebné analyzovať programovacie jazyky Java a Python hlavne zo syntaktickej stránky, ale taktiež z pohľadu ich fungovania a vlastností. Táto analýza je prezentovaná v podkapitole Porovnanie programovacích jazykov, ale taktiež v návrhu a implementácií.

Syntéza

Informácie a znalosti získané zo spomenutých analýz sme využili pri syntéze. Syntéza je popísaná v časti softvérového návrhu. V návrhu je navrhnuté spojenie ANTLR a PS Diagramu pri importe zdrojového kódu, kde sa z objektov typu Parse Tree, ktoré pri svojom vnútornom fungovaní vytvára softvérový nástroj ANTLR, vytvárajú objekty typu Flowchart, ktoré sú potrebné pre vykreslovanie interaktívnych vývojových diagramov programu PS Diagram. Bolo nutné použiť informácie z analýzy PS Diagramu, analýzy

ANTLR, analýzy gramatických pravidiel jazykov Java a Python ale taktiež používateľských požiadaviek.

Rešerš

Pre realizáciu tejto práce bol taktiež vykonaný rešerš literatúry a internetových zdrojov, zahraničných aj domácich. Rešerš sa týkal algoritmov a dátových štruktúr ale taktiež ich dôležitosti pri výučbe informatiky Páni Robert Sedqewick, Kevin Wayne rozoberajú vo svojej online dostupnej publikácií Algorithms 4th edition základné pojmy tvorby a optimalizácií algoritmov, pričom zdôrazňujú, že každý programátor potrebuje tieto znalosti mať. Pre hlbšie pochopenie rozpoznávania jazyka nám pomohla oficiálna dokumentácia ANTLR ale tiež kniha The Definitive ANTLR 4 Reference, ktorú napísal professor informatiky na univerzite v San Francisco Terrence Parr, pôvodný autor ANTLR. Pri analýze PS Diagramu pomohla officiálna príručka PS Diagram ako aj konzultácie s Miroslavom Bartyzalom. Pre analýzu programovacích jazykov Java a Python sme vychádzali z officiálnych dokumentácií týchto programovacích jazykov.

Metodika práce

Predmetom skúmania tejto práce sú algoritmy a ich vizualizácia pre zrýchlenie a skvalitnenie návrhu a implementácie procesov všeobecne. Riešenie tejto problematiky je inšpirované a podložené viacerými zahraničnými ale aj domácimi vedeckými a výskumnými prácami, ktorých predmetom skúmania sú algoritmy, dátové štruktúry, vývojové diagramy alebo prostriedky pre prekladanie a vizualizáciu algoritmov. Zdrojmi informácií sú aj oficiálne webové stránky a dokumentácie ANTLR (Another Tool for Language Recognition), PS Diagram, Java a Python. Ďalším zdrojom informácií sú prednášky a semináre absolvované počas štúdia, ktoré sa zaoberali problematikou algoritmov, vývojových diagramov a nástrojov na ich vývoj.

V úvodných častiach tejto práce sú uvedené teoretické poznatky z oblasti algoritmov a dátových štruktúr. Pri spracovaní tejto časti sú hlavne využité poznatky zo štúdia odbornej literatúry a jej analýzy. Získané vedomosti sú parafrázované a zhrnuté do stručnej podoby najdôležitejších skutočností týkajúcich sa problematiky algoritmov z hľadiska ich dopadu na spoločnosť a rôzne vedné odbory, ako aj dôležitosti ich štúdia.

V záverečnej časti sú prezentované výsledky bakalárskej práce, ktoré vznikli pomocou analýzy, návrhu a implementácie. Analýza je vykonaná z hľadiska užívateľských

požiadaviek a z hľadiska softvérových požiadaviek a obmedzení. Návrh je vykonaný podľa princípov softvérového návrhu s použitím UML diagramov a technickej špecifikácie. Implementácia je vykonaná v Eclipse IDE, s použitím Javy a ANTLR. Preverenie zdrojového kódu, testovanie funkčnosti, cloudové úložisko a zlučovanie zdrojového kódu s pôvodnou verziou programu prebieha pomocou GitHub v spolupráci s pôvodným autorom programu PS Diagram.

2.2.1 Rozpoznávanie jazyka a ANTLR

ANTLR (ANother Tool for Language Recognition) je softvérový nástroj používaný na rozpoznávanie jazyka. ANTLR je softvér s otvorenou licenciou, jeho pôvodný autor je Terence Parr, profesor informatiky na univerzite v San Franciscu. Tento nástroj je široko používaný a podporovaný, stále má však svoje nedostatky a stále je vo vývoji ako popisuje oficiálna dokumentácia. [7]

Twitter používa ANTLR na rozpoznávania vyše dvoch miliárd požiadaviek denne. Dátové sklady a systémy pre analýzu projektu Apache Hadoop, používajú ANTLR. Lex Machina používa ANTLR na extrahovanie informácií z právnických dokumentov. Oracle používa ANTLR pre svoje SQL IDE a nástroje migrácie dát. NetBeans IDE používa ANTLR pre C++. Hibernate rozpoznáva význam príkazov v jazyku HQL pomocou ANTLR. Okrem týchto veľkých a známych softvérových projektov sa ANTLR používa aj v stovkách iných menej známych projektov. Na GitHube má v čase tvorby tejto záverečnej práce ANTLR 9,3 tisícov pozitívnych hodnotení a 2,2 tisíc rôznych vývojových vetiev. [7]

ANTLR je platformovo a jazykovo nezávislý, dá sa použiť pri ľubovoľnom programovacom jazyku a môžeme pomocou neho spracovávať text z ľudských alebo programovacích jazykov ako aj z binárnych súborov.[8]

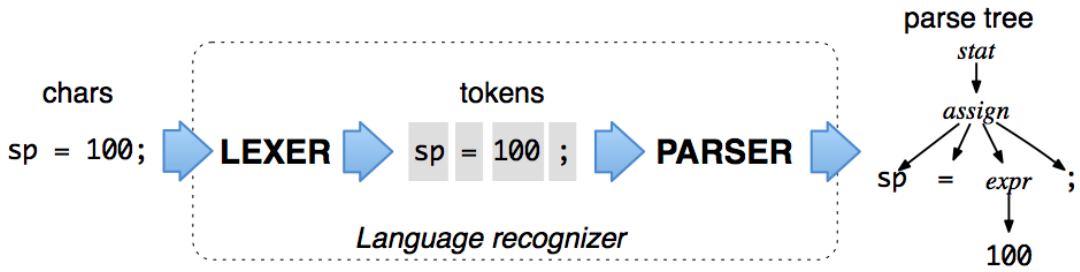
Softvérový balík s otvorenou licenciou obsahuje množstvo vstavaných gramatických súborov. Gramatický súbor je súbor, s príponou .g4 alebo .g pre staršie verzie, ktorý obsahuje gramatické pravidlá pre syntax a sémantiku daného jazyka, napísané v špecifickom jazyku ANTLR. Medzi dostupnými gramatickými súbormi nájdeme aj jazyky Java 8 a Python 3. [10]

ANTLR generuje objekty typu Parser, Lexer a Visitor, ktoré následne môže aplikácia používať. Lexer slúži na rozpoznávanie textu a vytváranie prúdu tokenov, Parser prijíma tokeny a vyvára Parse Tree (stromovú dátovú štruktúru, obsahujúcu rozpoznané

objekty jazyka). Visitor príjme Parse Tree a prechádza zaradom cez všetky uzly stromovej dátovej štruktúry a vykonáva programátorom definovanú aplikačnú logiku. [8][10]

ANTLR bol používaný v PS Diagrame aj pred našou bakalárskou prácou, na rozpoznávanie chýb pri tvorbe diagramov alebo pri importe a exporte diagramov do zdrojového kódu jazyka Pascal. [9]

V časti analýzy a návrhu je možné vidieť, že sme sa rozhodli použiť ANTLR pre rozpoznávanie textu zdrojových kódov jazyka Java aj Python. Použili sme na to dostupné gramatické balíky s miernymi úpravami pre potreby PS Diagramu. [7]



Obrázok č.2 **ANTLR Lexer, Parser a Parse Tree**

Zdroj:https://www.researchgate.net/figure/Flow-of-ANTLR-generated-Lexers-and-Parsers_fig4_318477435

Pre bližšie znázornenie vytvoreného prekladača sú v nasledujúcich kapitolách použité niektoré vybrané diagramy jazyka UML. Unified Modeling Language alebo UML je v softvérovom inžinierstve grafický jazyk na vizualizáciu, špecifikáciu, navrhovanie a dokumentáciu programových systémov. [22] Medzi štandardné diagramy UML patria use case diagram, activity diagram, sekvenčný diagram a pod. V práci je použitý *Use Case Diagram* na znázornenie používania programu PS Diagram, *Class Diagram*, teda diagram tried, ktorý znázorňuje triedy a ich vzťahy a sekvenčný diagram, ktorý znázorňuje volanie a spätné reťazenie pri procese importu zdrojového kódu.

3. Výsledky práce

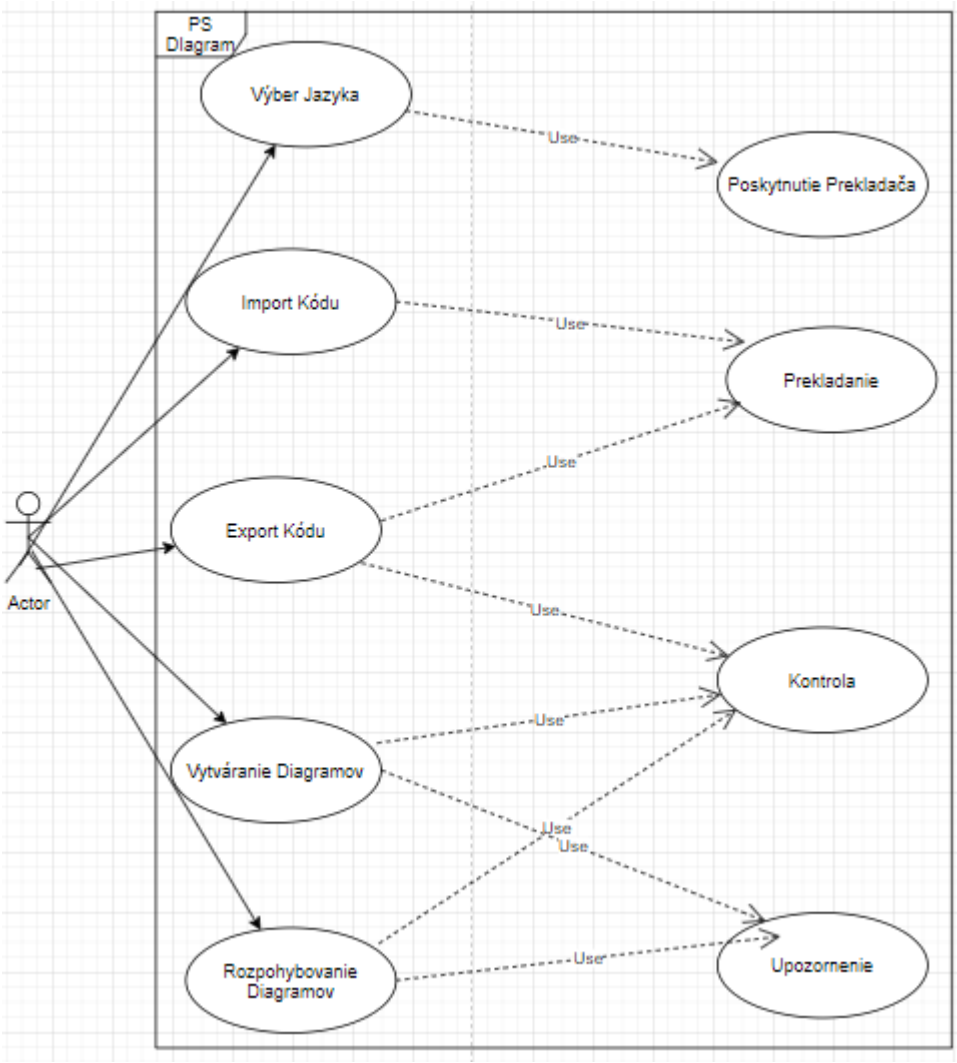
Realizácia tejto bakalárskej práce sa začala analýzou požiadaviek a analýzou PS Diagramu, ktorý bude rozšírený o novú funkcionalitu. Z výsledku analýzy vznikol návrh, ktorý je následne implementovaný.

3.1 Analýza

Nasledujúca analýza je rozdelená na dve časti, analýzu požiadaviek, resp. analýza možného prínosu pre koncového používateľa softvéru, a tiež softvérová analýza, resp. analýza už existujúceho softvéru, PS Diagram.

3.1.1 Analýza požiadaviek

Požiadavka z používateľského hľadiska pre túto prácu je pridať do programu PS diagram možnosť pre používateľa si vybrať jazyk Python alebo Java pri importe alebo exporte zdrojového kódu. Pre konzistenciu a kontinuitu s doterajším fungovaním programu PS Diagram je potrebné pri návrhu a implementácií dodržať obmedzenia a pravidlá, ktoré platia pri module prekladania jazyka Pascal. Na *obrázku č.3* vidieť všeobecný prehľad fungovania modulu importu a exportu PS Diagramu, z používateľského hľadiska. [9]



Obrázok č.3 **Use Case Diagram** [vlastné spracovanie]

3.1.2 Softvérová analýza

Okrem analýzy požiadaviek tejto práce je potrebné vykonať aj analýzu programu PS Diagram, zo softvérového hľadiska.

Proces analýzy je vykonaný získavaním informácií od pôvodného autora softvéru ale aj z dokumentácie a samotného zdrojového kódu. [6][9]

Program je vytvorený z viacerých modulov, z ktorých budeme meniť modul Import/Export a do modulu prekladania jazykov budeme pridávať nové prekladače, resp. softvérové komponenty, ktoré sú schopné prekladať programovacie jazyky do vývojových diagramov a naopak.[9]

Pri návrhu a implementácií je potrebné dodržať všetky obmedzenia, ktoré platia pre modul prekladača jazyka Pascal, ale treba taktiež vziať do úvahy rozdielny charakter týchto jazykov.

Zo všeobecného hľadiska budú platiť nasledovné obmedzenia:

1. Je možné prekladať iba telo jednej metódy
2. Prekladač podporuje štandardné dátové typy a štandardnú syntax daného programovacieho jazyka
3. Prekladač dodrží logiku cyklov, procesov a podmienok
4. Prekladač upozorní používateľa ak zadá nesprávne vstupy
5. Všetky prvky vývojového diagramu je možné preniesť do programovacieho jazyka
6. Z príkazov programovacieho jazyka je možné preložiť do vývojových diagramov iba tie, ktoré sú vykonateľné v rozmedzí PS Diagram, cykly, podmienky, premenné, aritmetické, logické a binárne operácie, porovnávanie, vstup a výstup na konzolu (rôzne knižničné funkcie alebo objekty nie sú podporované)
7. Prekladač upozorní používateľa na dodatočné úpravy zdrojového kódu, ktoré je potrebné vykonať aby bol zdrojový kód funkčný. [9]

Jazyk Python patrí medzi netypové (dynamically typed languages) a Java medzi typové (statically typed languages) jazyky. To znamená, že pred použitím premenných v jazyku Java je potrebné najprv uviesť ich dátový typ, v Pythone je dátový typ určovaný pri behu program podľa hodnôt ktoré premenná nadobúda. Pri tvorbe vývojových diagramov v PS Diagrame sa však dátové typy neuvádzajú preto v prípade Javy bude treba typy určovať implicitne kým pri Pythone tento problém nemusíme brať do úvahy. [14][5]

Pri importe zdrojového kódu bude potrebné rozpoznať jednotlivé príkazy z textu zdrojového kódu daného jazyka a vytvoriť objekty reprezentujúce vývojový diagram so správnymi hodnotami a usporiadané v správnom poradí.

Pri exporte zdrojového kódu bude potrebné extrahovať informácie o hodnotách, typoch a postupnosti procesov z objektov reprezentujúcich vývojový diagram a vytvoriť príkazy programovacieho jazyka so správnymi hodnotami a v správnom poradí.

Program PS Diagram je vytvorený v jazyku Java.[9]

3.2 Návrh

Návrh je zostavený z informácií vyplývajúcich z analýzy požiadaviek. Cieľom tohto návrhu je transformovať požiadavky do jednoducho implementovateľnej formy.

3.2.1 Špecifikovanie problémov a úloh

Je potrebné jasne definovať problémy, ktoré vyplývajú z požiadaviek vytvorených pri analýze v predchádzajúcej kapitole. Následne je možné rozdeliť problémy do konkrétnych úloh, pričom využijeme dobre známy princíp *dekompozície*.

Problémy a úlohy platia rovnako pre jazyk Java aj jazyk Python, okrem niektorých výnimiek, ktoré sú uvedené pri definícií problému.

- Prekladanie
    - Prekladanie z programovacieho jazyka do PS Diagramu
        - Rozpoznávanie jazyka
            - Definícia syntaxe zrozumiteľnej pre program
            - Definícia sémantiky jazyka
            - Voľba správnej dátovej štruktúry na uchovávanie extrahovaných údajov
            - Kontrola a extrahovanie príkazov jazyka
            - Uloženie údajov do dátových štruktúr
            - Prechod do Generácie diagramových objektov
        - Generácia diagramových objektov
            - Usporiadanie údajov do formy prijateľnej pre modul zobrazenia vývojových diagramov
            - Odoslanie údajov do modulu zobrazenia vývojových diagramov
    - Prekladanie z jazyka PS Diagram do programovacieho jazyka
        - Extrahovanie údajov z objektov diagramu
            - Extrahovanie typu objektu, hodnoty objektu
            - Odvodzovanie dátových typov (platí iba pre typové jazyky)
            - Extrahovanie vzťahov objektu voči okoliu
        - Kontrola údajov
            - Kontrola výskytu chýb

        - Vytvorenie chybových oznámení a varovaní
    - Generovanie príkazov zdrojového kódu do textového formátu
        - Transformácia jednotlivých hodnôt do textu zdrojového kódu
        - Usporiadanie jednotlivých príkazov
    - Odoslanie textu zdrojového kódu do modulu pre zobrazenie zdrojového kódu

3.2.2 Návrh softvérového riešenia

Pri návrhu softvérového riešenia je navrhnuté riešenie na problémy vyplývajúce z analýzy, ktoré sú následne implementované. Problémy sú hierarchicky usporiadané a podľa stupňa komplexnosti môžu predstavovať moduly alebo metódy. Jeden modul resp. metóda môže riešiť viacero čiastkových problémov a takisto na riešenie jedného problému môže byť navrhnutých viacero tried, resp. metód.

Pri návrhu sú definované dátové štruktúry, metódy a triedy, následne moduly, v ktorých sa triedy nachádzajú a nakoniec ich prepojenie na ostatné moduly a triedy, podľa *stratégie zdola nahor*, ktorá je vhodnejšia pri už existujúcom systéme.

Komponenty, dátové štruktúry, metódy a rozhrania sú v návrhu abstraktné, abstrahuje sa od ich konkrétnych implementácií.

Dátové štruktúry

Stromová dátová štruktúra *Diagram {typ; hodnota; predchadzajuciDiagram, vnoreneDiagramy}*

Táto dátová štruktúra je hierarchicky usporiadaná z uzlov a ukazovateľov na predchádzajúci (rodičovský) Diagram.[9] Každá inštancia tejto dátovej štruktúry reprezentuje časť vývojového diagramu. Diagram môže mať vnorené diagramy, napr. *cyklus for* má v svojom tele ďalšie príkazy a po ňom nasledujú ďalšie, ktoré už nepatria do jeho tela a týmto spôsobom ich je možné odlíšiť, podľa hĺbky resp. hierarchickej úrovne (či sú *vnorené* alebo *nasledujúce* elementy). [9]

Diagram môže nadobúdať rôzne typy podľa toho aký objekt reprezentuje vo vývojovom diagrame napr. kosoštvorec bude typu DECISION, obdĺžnik typu PROCESS a pod. [9]

Metódy

generujZdrojovyKod(**Input**: objekty diagramov; **Output:** text zdrojového kódu);

generujDiagram(**Input:** text zdrojového kódu: **Output:** objekty diagramov);

nastavPremenne(**Input**: objekty diagramov; **Output:** zoznam premenných);

nastavTypPremennej(**Input:** premenna; **Output:** typ a premenna);

nastavCyklus(**Input:** diagram-cyklus; **Output:** zdrojový kód);

nastavRozhodovanie(**Input:** diagram-decision **Output:** zdrojový kód);

skontrolujKod(**Input:** zdrojovy kod; **Output**: chyby ak existuju);

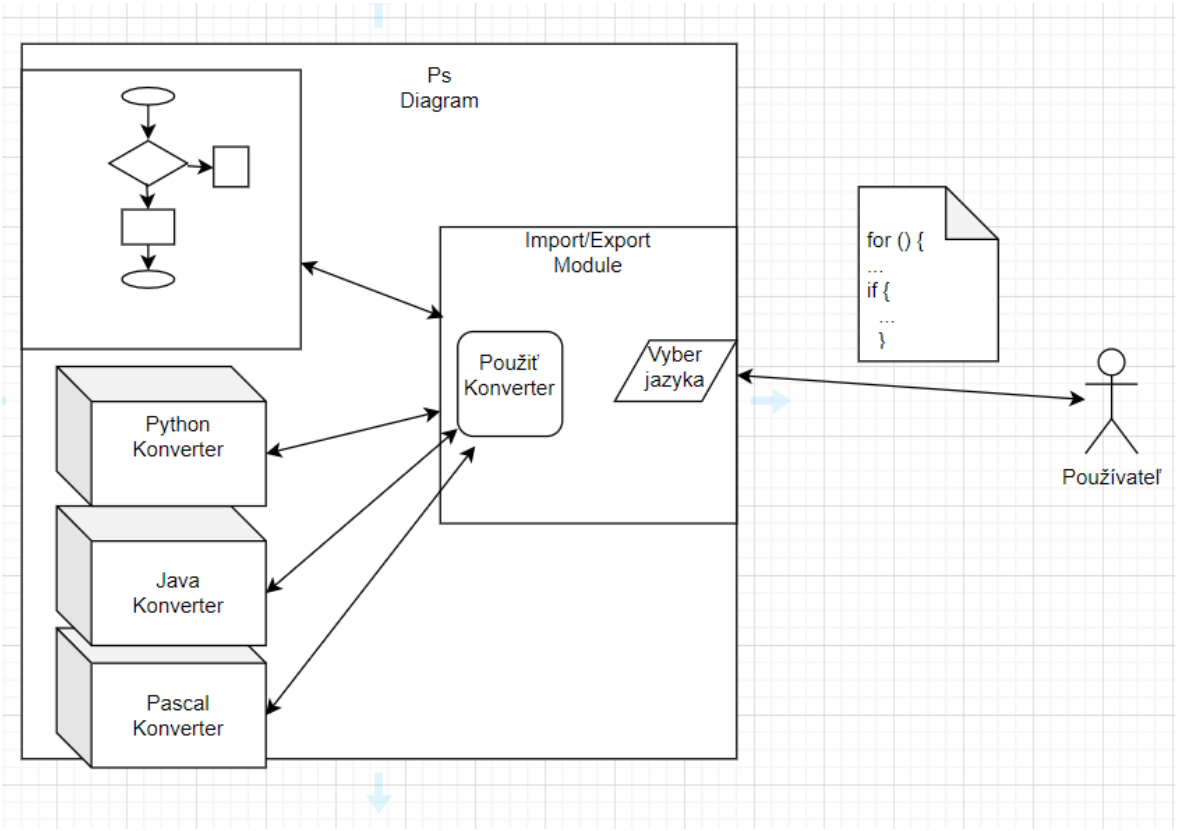
vlozVnorenyDiagram(**Input:** Diagram, rodicovsky Diagram; **Output**: Diagram);

vlozDalsiDiagram(**Input:** Diagram, predchadzajuci Diagram: **Output**: Diagram);

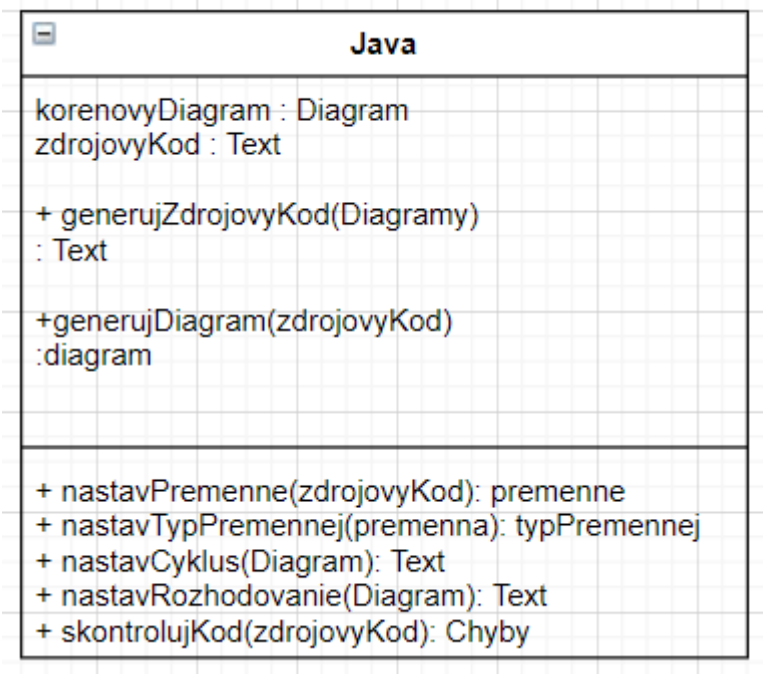
Triedy

Triedy Java a Python sú podobné, obe majú dve verejné metódy *generujZdrojovyKod* a *generujDiagram*, ktoré pomocou ostatných *súkromných* metód zabezpečujú prekladanie, podľa vzoru triedy Pascal. [9]

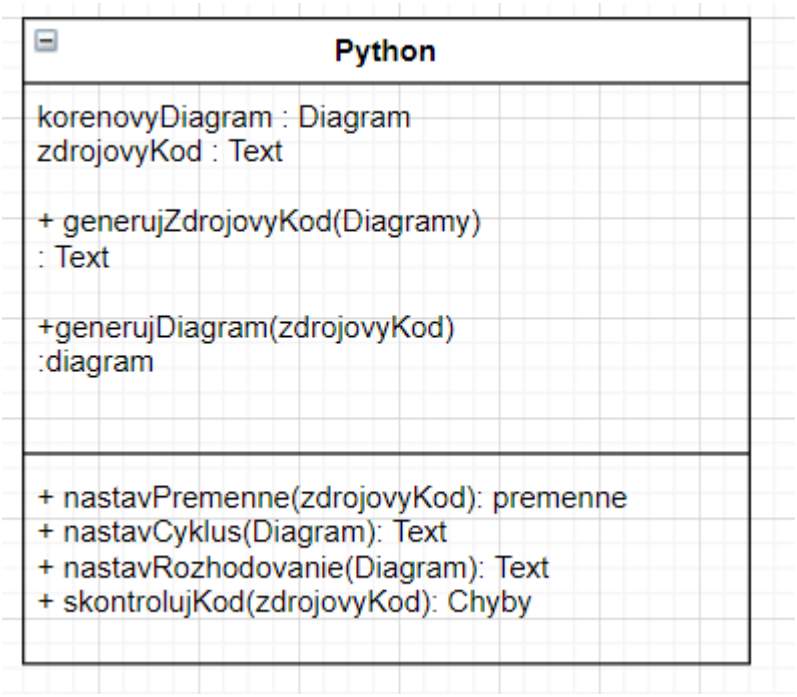
Vnútorná logika týchto tried je skrytá pred ich používateľmi, preto sa navonok používajú rovnako a to volaním metód *generujZdrojovyKod*, ktorá sa použije pri exporte zdrojového kódu alebo *generujDiagram*, ktorá sa použije pri Importe zdrojového kódu.



Obrázok č.4 **Architektúra programu PS Diagram** [vlastné spracovanie]

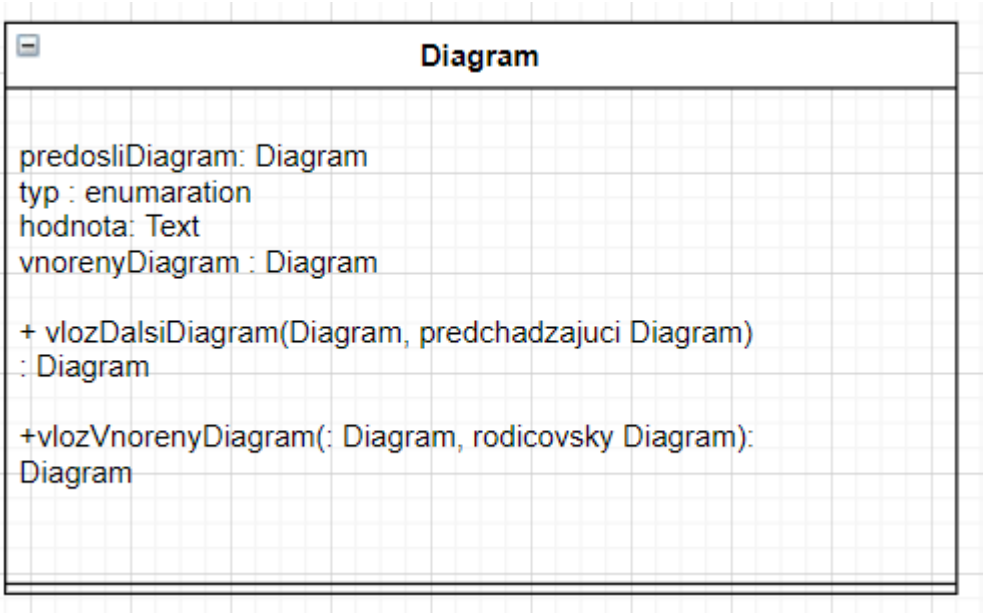


Obrázok č.5 **Trieda Java** [vlastné spracovanie]



Obrázok č.6 **Trieda Python** [vlastné spracovanie]

Trieda Python neobsahuje súkromnú metódu *nastavTypPremennej* lebo pri prekladaní do a z jazyka Python nenastavujeme dátové typy premenných.



Obrázok č.7 ***Dátová štruktúra Diagram*** [vlastné spracovanie]

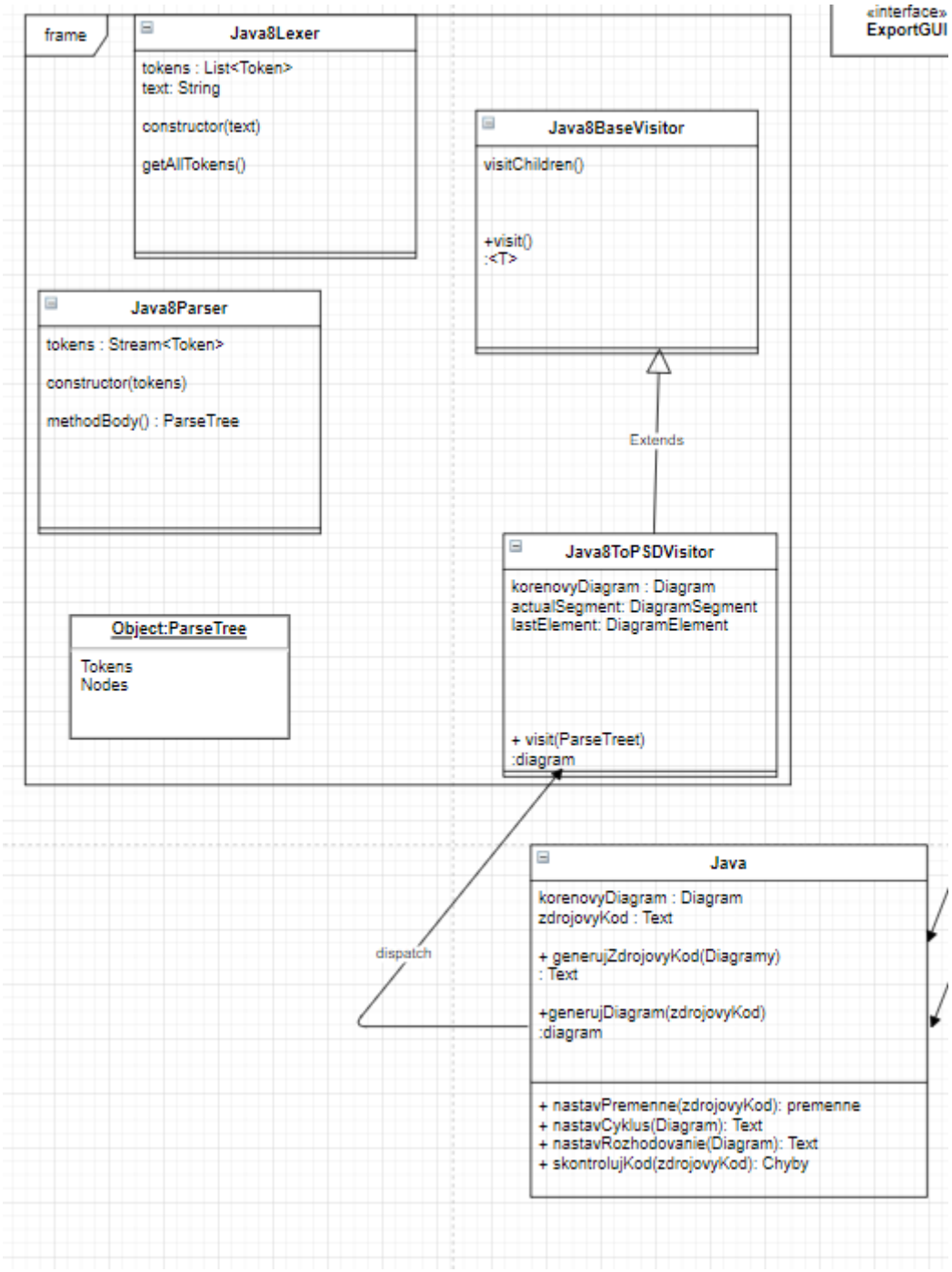
Usporiadanie tried

Metódu *generujDiagram* triedy Python a Java delegujú modulu *ANTLR* (*Java Lexer* a *Python Lexer*), ktorý podľa nami definovaných gramatických pravidiel rozpozná zdrojový kód a vytvára nami definované dátové štruktúry (*Diagramy*). [10][11]

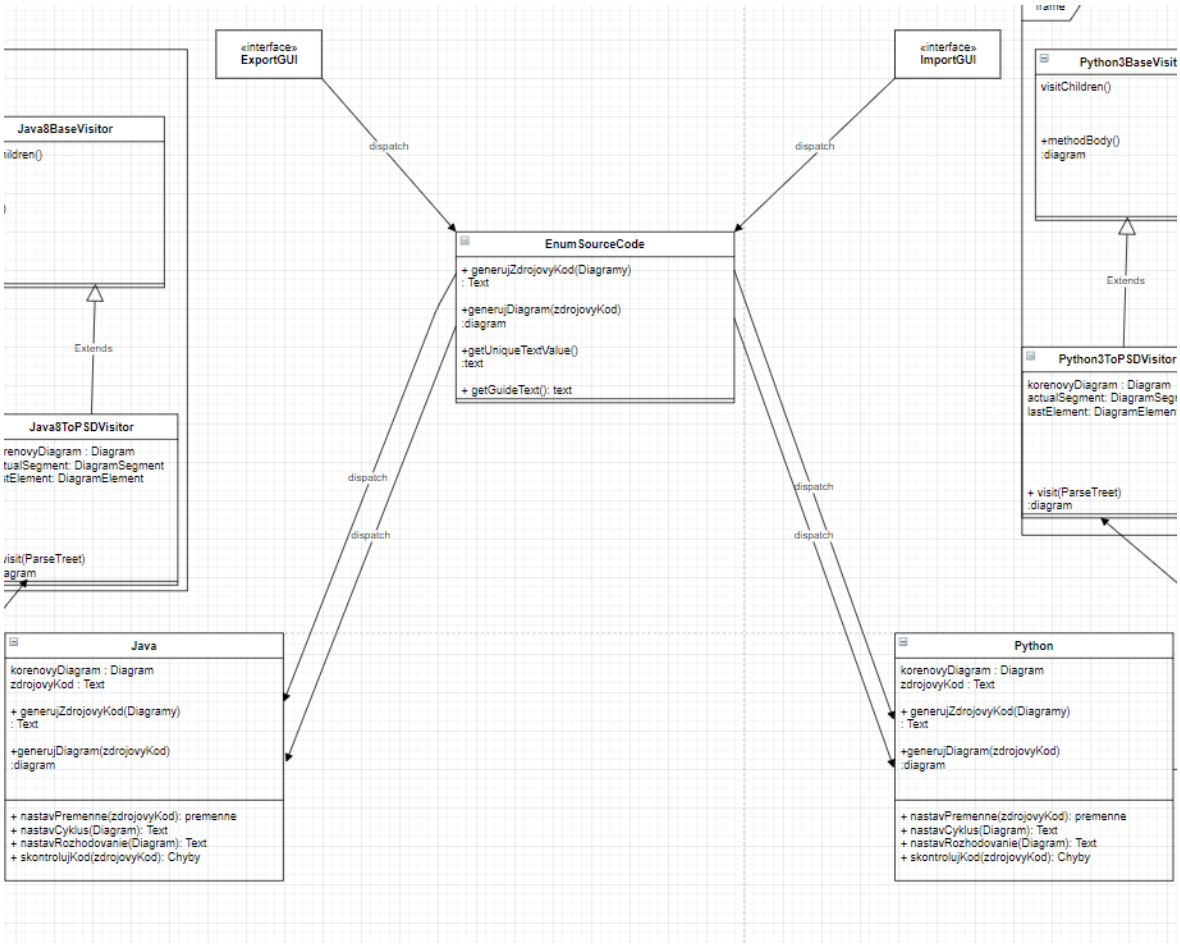
Nateraz je od vnútorného fungovania týchto modulov abstrahované, detailnejšie sú opísané v implementácií.

Do modulu Import/Export sú pridané nové možnosti pre použitie jazykovej triedy, doteraz tam bol iba Pascal, a navrhnuté sú Java a Python. Modul Import/Export vyberie triedu na parsovanie, resp. prekladanie podľa používateľom zvoleného programovacieho jazyka a bude volať metódy *generujZdrojovyKod* pre export alebo *generujDiagram* pre import.

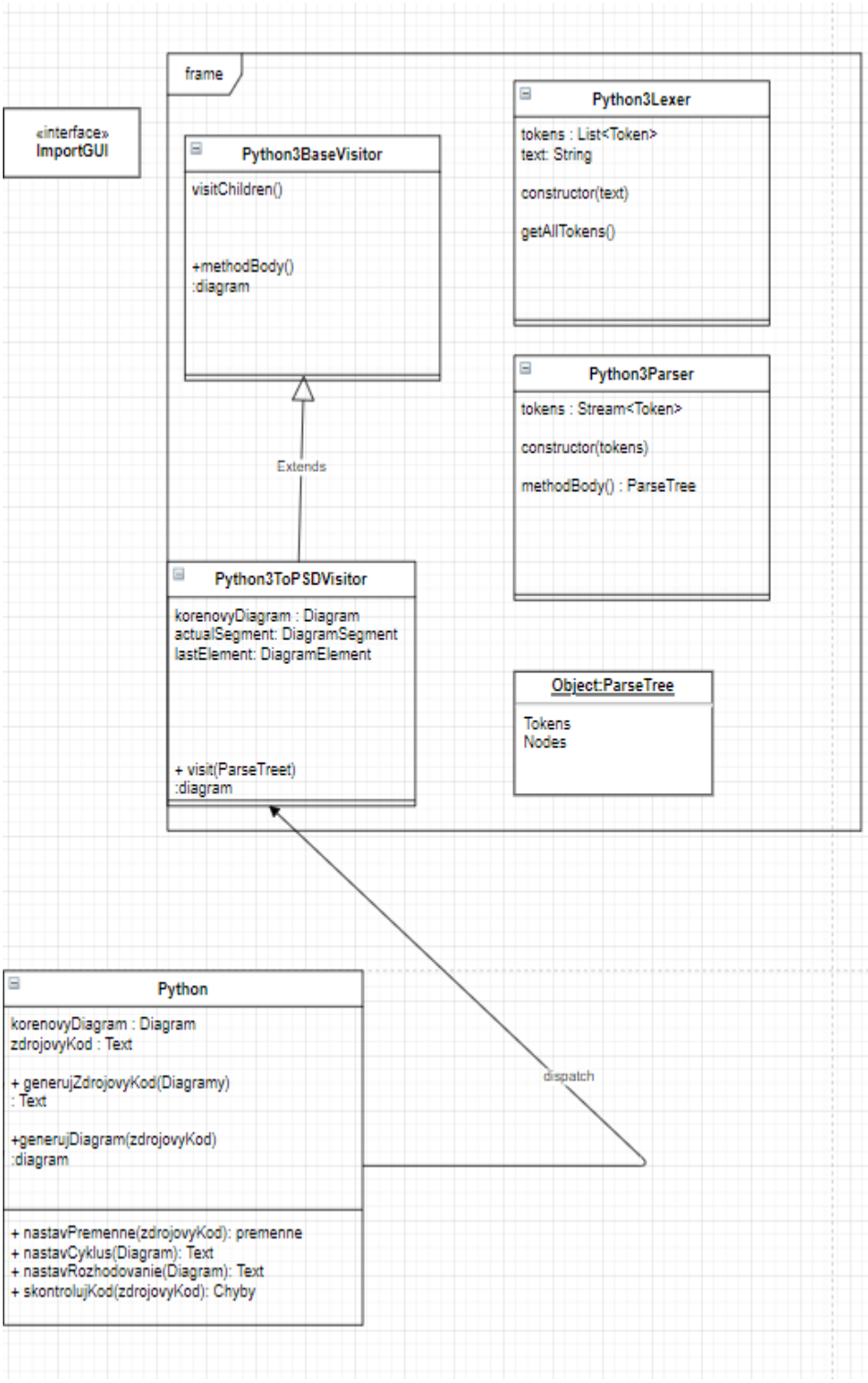
Na nasledujúcich diagramoch tried UML je možné vidieť triedu *EnumSourceCode*, trieda obsahuje enumeráciu PASCAL, JAVA a PYTHON a je používaná grafickým rozhraním. Trieda *EnumSourceCode* deleguje fungovanie vlastných metód *getSourceCode* a *getDiagram* pri importovaní a exportovaní zdrojového kódu triedam Java a Python (od triedy Pascal je abstrahované pre prehľadnosť). Triedy Python a Java pri importovaní zdrojového kódu delegujú túto úlohu rozhraniu vytvorenému pomocou tried ANTLR. Rozhranie pre rozpoznávanie jazyka Java, resp. Python vytvára objekt typu *ParseTree* pomocou tried *Lexer* a *Parser*. Trieda *Java8ToPSDVisitor* dedí od triedy *Java8BaseVisitor* a preťažuje jej generický dátový typ <*T*> dátovým typom diagram, generická metóda *visit(ParseTree)* vracia objekt typu *Diagram*. V triedach, ktoré používajú rozhranie ANTLR je do značnej miery abstrahované od vnútorného fungovania rozhrania. Pri vytváraní objektov typu *Lexer* a *Parser* poskytnú text, a z metódy *visit(ParseTree)* získajú objekt typu *Diagram*. (Nasledujúci diagram tried je rozdelení do troch obrázkov, kvôli viditeľnosti.)



Obrázok č.8 **Java Class Diagram** [vlastné spracovanie]



Obrázok č.9 **EnumSourceCode Class Diagram** [vlastné spracovanie]



Obrázok č.10 **Python Class Diagram** [vlastné spracovanie]

Navrhnuté softvérové riešenie je implementované v programovacom jazyku Java, keďže program PS Diagram, ktorý je predmetom rozšírenia, je implementovaný v tomto jazyku. Použitý je aj softvérový nástroj ANTLR (Another Tool for Language Recognition) na rozpoznávanie textu zdrojových kódov daných programovacích jazykov.

Java a ANTLR predstavujú dve nosné technológie tohoto programu a práve preto sú použité na rozšírenie programu.

Pri implementácií je zachované anglické názvoslovie, kvôli kontinuite s názvoslovím použitým v programe.

3.3 Implementácia

Implementáciou dátovej štruktúry Diagram je trieda FlowChart, ktorá používa XML anotácie a údaje v nej skladované sú usporiadané hierarchicky podobne ako v XML. Táto trieda, už bola v pôvodnom balíku a je ideálny kandidát implementácie dátovej štruktúry *Diagram*. Táto trieda je pri implementácií často používaná.

@XmlRootElement(name = "flowchart")

@XmlAccessorType(XmlAccessType.NONE)

@XmlType(name = "flowchart")

public final class Flowchart<S extends FlowchartSegment<S, E>, E extends FlowchartElement<S, E>>

implements Iterable<S> [9]

Metóda *getFlowchart* deleguje extrahovanie triede *JavaToPsdVisitor,* ktorá je špecifickou implementáciou triedy *Java8BaseVisitor* z ANTLR modulu pre rozpoznávanie syntaxe jazyka Java.[10][11]

ParseTree tree = parser.methodBody();

if (tree!=null) {

JavaToPSDVisitor visitor = new JavaToPSDVisitor();

Flowchart<LayoutSegment, LayoutElement> flowchart= visitor.visit(tree);

return flowchart; } [vlastné spracovanie]

V tele metódy sa vytvorí strom (*ParseTree*), ktorého uzly sú jednotlivé príkazy alebo časti príkazov, hierarchicky usporiadané. [8][vlastné spracovanie]

Objekt typu *JavaToPSDVisitor* prechádza všetkými uzlami pod uzlom *methodBody*, čiže len od tela metódy do hlbšieho detailu, pričom pri vložení kódu širšieho ako telo metódy, ktorý obsahuje napr. niekoľko ďalších metód alebo premenné danej triedy a pod. import kódu skončí s chybovou hláškou. [10][vlastné spracovanie]

Preto aj pri vkladaní zdrojového kódu pre import je ako aj v pôvodnom Pascal module aj pri Jave a Pythone upozornenie, že je potrebné vložiť iba telo jednej metódy. [9]

JavaToPSDVisitor extends Java8BaseVisitor<Flowchart<LayoutSegment, LayoutElement>>

Java8BaseVisitor<T> extends AbstractParseTreeVisitor<T> implements Java8Visitor<T>

Vidíme, že generický dátový typ *<T>* bol nahradený dátovým typom *FlowChart*, ktorý reprezentuje dátovú štruktúru *Diagram*. Teda táto trieda bude pracovať s objektami triedy *FlowChart*. [vlastné spracovanie]

JavaToPSDVisitor obsahuje metódy, ktoré sú volané vtedy, keď v strome (ParseTree) objekt typu *Visitor* prechádza konkrétnym uzlom. Tieto metódy môžeme, ale nemusíme preťažiť (*@Override*) a implementovať do nich vytváranie a vnáranie uzlov typu *FlowChart*, alebo prípadne ľubovoľnú aplikačnú logiku.[8]

```
@Override
    public Flowchart<LayoutSegment, LayoutElement> visitIfThenElseStatement(Java8Parser.IfThenElseStatementContext ctx) {
        String condition = "true";
        if (ctx.getChildCount() > 0) {
            for (ParseTree child : ctx.children) {
                if (child instanceof StatementContext) {
                    visitChildren(((StatementContext)child));
                }
                if (child instanceof StatementNoShortIfContext) {
```

visitChildren(((StatementNoShortIfContext)child));

}

if (child instanceof ExpressionContext) {

condition = child.getText();

visitChildren(((ExpressionContext)child));

}

}

}

lastElement = actualSegment.addSymbol(lastElement, EnumSymbol.DECISION.getInstance(condition));

visitChildren(ctx);return flowchart;

} [vlastné spracovanie]

Táto metóda zabezpečuje vytvorenie *if then else* diagramu typu *DECISION* keď v strome vytvorenom časťami zdrojového kódu natrafí na *if then else* podmienku. Metóda volá *visitChildren* teda ešte pred dokončením metódy, *visitor* prechádza cez vnorené, resp. detské uzly. Keď už ďalší takýto uzol neexistuje vráti výsledok a potom aj táto metóda vráti výsledok metóde svojho rodičovského uzlu. Takýto agregovaný výsledok nakoniec vráti *visitor* ako odpoveď na volanie metódy *visitMethodBody*. V odpovedi bude objekt typu *Flowchart*, ktorý bude obsahovať všetky príkazy z danej metódy vo forme usporiadaných objektov vývojového diagramu, pripravených na zobrazenie.

Metóda *generujZdrojovyKod* je implementovaná pomocou Javy bez využitia dodatočných knižníc alebo softvérových komponentov. Používame pri tom regulárne výrazy a aplikačnú logiku, pomocou ktorých extrahujeme kód z objektov triedy *FlowChart,* a postupne pridávame a skladáme do textovej reprezentácie zdrojového kódu (*String).*

Toto je napríklad kód, ktorý podmienkovú časť výrazu rozdelí na časti okolo znamienka. (>,<,=,!= atď.)

String [] conditionParts = conditionString.split("\\>|\\<|\\<=|\\>=|=|!|!=");

Metódou *instanceof* zistíme, inštanciou ktorej z tried dediacich od *ParseTree* je daný objekt a následne podľa toho vytvoríme príkaz v danom programovacom jazyku.

else if (symbol ***instanceof*** *IO)*

Pomocou rôznych podmienok a regulárnych výrazov aj kontrolujeme validitu používateľom zadaných hodnôt. Rozlišujeme chybu a varovanie.

Chyba je vážnejšia a program pri nej nedokončí konverziu ale vypíše chybovú hlášku. Napr. keď používateľ zadá text namiesto čísla, alebo je vstup do metódy deformovaný iným spôsobom.

catch (NumberFormatException | HeadlessException e) {

JOptionPane.showMessageDialog(null,

*"<html>Zdrojový kód se nepodařilo vytvořit!
problémový symbol vlastní popisek: \"" + symbol.getValue() + "\".</html>",*

"Chyba při generování zdrojového kódu", JOptionPane.ERROR_MESSAGE);

errored = true;

Varovanie je menej vážne a program dokončí konverziu a vypíše chybovú hlášku.

JOptionPane.showMessageDialog(null,

*"<html>Zdrojový kód byl vygenerován s následujícím upozorněním:
Některý symbol nemá vyplněnu svou funkci!</html>",*

"Nevyplněná funkce symbolu", JOptionPane.WARNING_MESSAGE);

Trieda *EnumSourceCode* je enumeračný dátový typ, ktorý poskytuje základné metódy na prácu s importom a exportom jednotlivých jazykov. V pôvodnej verzií programu táto trieda existovala a obsahovala jednu konštantu s názvom PASCAL, ktorej metódy *getFlowchart* a *getSourceCode* odkazovali na triedu Pascal, v ktorej je uzavretá aplikačná logika pre import a export pascalu. Trieda *Pascal* sa nachádza v rovnakom balíku ako trieda *EnumSourceCode*, teda nie je potrebné *Pascal* importov, metódy *getSourceCode*

a *getFlowchart* sú statické a verejné preto je možné implementáciu prekladania možné delegovať jednoduchým príkazom *Pascal.getSourceCode(code);* a vrátiť výstup tejto metódy triedy *Pascal*. Trieda *EnumSourceCode*, resp. jednotlivé konštanty tohto enumeračného dátového typu ponúkajú ešte dve metódy *getUniqueTextValue* a *getGuideText,* ktoré ponúkajú názov jazyka a pomocný text.

Rovnakým resp. veľmi podobným spôsobom sú implementované aj konštanty JAVA a PYTHON, ktoré rozširujú tento enumeračný dátový typ o nové jazyky.

JAVA

```
{
    @Override
    public Flowchart<LayoutSegment, LayoutElement> getFlowchart(String code)
    {
       return Java.getFlowchart(code);
    }

    @Override
    public String getSourceCode(Flowchart<LayoutSegment, LayoutElement> flowchart,
String name)
    {
       return Java.getSourceCode(flowchart, name);
    }

    @Override
    public String getUniqueTextValue()
    {
       return "Java";
    }

    @Override
    public String getGuideText()
```

```
    {
        return "vlozte len telo jednej funckie !! od '{' po '}' !!";
    }
},

    PYTHON
{
    @Override
    public Flowchart<LayoutSegment, LayoutElement> getFlowchart(String code)
    {
        return Python.getFlowchart(code);
    }

    @Override
    public String getSourceCode(Flowchart<LayoutSegment, LayoutElement> flowchart,
String name)
    {
        return Python.getSourceCode(flowchart, name);
    }

    @Override
    public String getUniqueTextValue()
    {
        return "Python";
    }

    @Override
    public String getGuideText()
    {
        return "vlozte len telo jednej funckie  ";
```

```
    }
};
```

Trieda *EnumSourceCode* je používaná grafickým rozhraním programu PS Diagram, a pre ponúknutie nových programovacích jazykov pre modul import/export postačuje pridať nové konštanty danej triedy. Je dobré poznamenať, že program PS Diagram bol teda navrhnutý tak, že je ho možné pomerne jednoducho rozširovať.

V telách metód triedy *JavaToPSDVisitor* program pracuje s dvoma dátovými štruktúrami a to so štruktúrou *Diagram* a štruktúrou *ParseTree*, resp. s objektami triedy *Flowchart,* ktorá je vlastná trieda programu PS Diagram a s objektami typu *ParseTree,* ktoré vygeneroval *Parser* a *Lexer* a prechádza nimi *Visitor*.

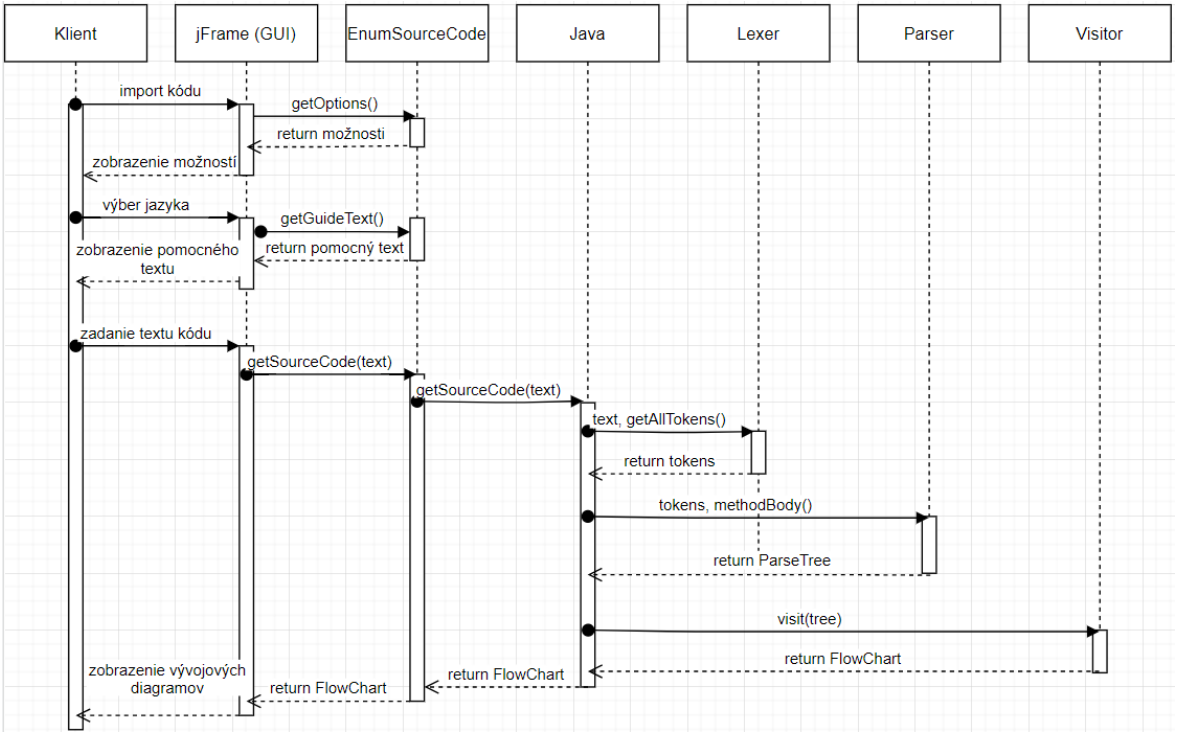
```
@Override
    public Flowchart<LayoutSegment, LayoutElement>
visitSwitchStatement(Java8Parser.SwitchStatementContext ctx) {
        LayoutElement tmpSwitch =  null;
        LayoutSegment tmpSegment = actualSegment;
        if (ctx.getChildCount() > 0) {
            for (ParseTree child : ctx.children) {
                if (child instanceof ExpressionContext) {
                    lastElement = actualSegment.addSymbol(lastElement,
EnumSymbol.SWITCH.getInstance(child.getText()), ctx.getChildCount());

    cz.miroslavbartyzal.psdiagram.app.gui.symbolFunctionForms.Switch.generateValu
es(lastElement, child.getText(), new String[0]);
                    tmpSwitch = lastElement;
                }
                if (child instanceof SwitchBlockContext) {
                    visitSwitchBlock((SwitchBlockContext) child);
```

```
                    }
                }
            }
            lastElement = tmpSwitch;
            actualSegment = tmpSegment;
            return flowchart;
        }
```

Objekty typu *ParseTree* poskytujú informáciu o počte detských uzlov metódou *getChildCount* a metódou *children* poskytnú tieto detské uzly. Pomocou *instanceof* je možné zistiť, o ktorý typ detského uzla ide. Pokiaľ ide o *ExpressionContext* vo vnútry tejto metódy, ktorá pracuje so *SwitchStatementContext* a teda detský uzol *SwitchStatementContext* typu *ExpressionContext* je možné usúdiť, že ide o premennú používanú pri vyhodnocovaní jednotlivých vetiev *switch*. Pomocou príkazu *lastElement=actualSegment.addSymbol* je pridaný ako posledný element nový vývojový diagram reprezentujúci začiatok *switch*. Pokiaľ ide o *SwitchBlockContext* je možné usúdiť, že ide o vetvu *switch,* všetky tieto vetvy sú vnorené do segmentu, ktorý je o jednu úroveň nižšie v strome ako uzol začiatku *switch* a uzlov, ktoré budú sekvenčne nasledovať, teda ide o detské uzly *switch*. Následne sa *actualSegment* vráti na pôvodnú úroveň a *lastElement* sa nastaví na *switch* ako celok.

Implementácia sa teda skladá zo základných častí spomenutých v tejto kapitole a to sú:

- Využívanie objektov typu *Flowchart* pre vývojové diagramy
- Využívanie objektov typu *ParseTree* pre rozpoznávanie jazyka
- Implementovanie metód pri navštevovaní jednotlivých uzlov *ParseTree*
- Ponúknutie nových konštánt reprezentujúcich nové programovacie jazyky pre import a export v triede *EnumSourceCode* a to JAVA a PYTHON
- Implementácia exportu zdrojového kódu pomocou *RegExp* a aplikačnej logiky v triedach *Java* a *Python*



Obrázok č.11 **Sekvenčný Diagram** [vlastné spracovanie]

Na sekvenčnom diagrame môžeme vidieť priebeh procesu importu zdrojového kódu, jednotlivé objekty a spôsob akým si volaním metód posielajú údaje a delegujú úlohy. Klient iniciuje import kódu, objekt *jFrame* (Swing) reprezentujúci grafické rozhranie volá metódu *getOptions()* triedy *EnumSourceCode*, ktorá vracia všetky dostupné programovacie jazyky na prekladanie. Klient si vyberie programovací jazyk, táto informácia je uložená v grafickom rozhraní a podľa nej sa vyberie správny programovací jazyk, a pre vybraný programovací jazyk, resp. konštanty enumeračného dátového typu *EnumSourceCode* sa zavolá metóda *getGuideText()*, ktorá vráti objektu *jFrame* pomocný text, ktorý následne klient môže vidieť zobrazený v grafickom rozhraní programu. Klient zadá text, ktorý putuje cez *jFrame*, *EnumSourceCode* až do objektu typu *Java (*v prípade, že klient vybral tento programovací jazyk). V *Jave* je vytvorený objekt typu *Lexer*, do konštruktora prijíma parameter *text*, a je volaná metóda, *getAllTokens(),* ktorá vráti prúd tokenov. Následne je vytvorený objekt typu *Parser*, do konštruktora prijíma prúd tokenov a volanie metódy *methodBody()*, vráti objekt typu *ParseTree*, ktorý reprezentuje hierarchicky usporiadané rozpoznané príkazy daného programovacieho jazyka. Objekt typu *Visitor* príjme do metódy *visit(parseTree)* vytvorený rozpoznávací strom, a vráti objekt typu *FlowChart,* obsahujúci interaktívne vývojové diagramy, ktoré je PS Diagram schopný vykresliť v grafickom rozhraní.

Implementácia prešla fázami testovania aj *code review* v spolupráci s autorom pôvodného programu, Miroslavom Bartyzalom, a je v procese zlučovania s oficiálnou verziou programu PS Diagram, ktorý bol po minulé roky stiahnutý približne 5000 krát ročne.

Záver a diskusia

V tejto bakalárskej práci je prezentované vykonanie analýzy, návrhu a čiastočnej implementácie prekladania zdrojového kódu jazykov Python a Java. Počas analýzy sú popísané požiadavky, a abstraktný prehľad softvéru PS Diagram, ktorý je predmetom rozšírenia funkcionality.

Z požiadaviek získaných počas analýzy je vytvorená špecifikácia pre implementáciu. Pritom sú použité princípy softvérového návrhu a to : dekompozícia, dátová, technická a aplikačná abstrakcia, a stratégia zdola nahor.

Implementácia je vykonaná v Jave v kombinácií s ANTLR. Sú implementované metódy, triedy a dátové štruktúry z návrhu, a aj dodržané podmienky a obmedzenia z analýzy.

Pri implementácií sa často využívajú generiká (*Java Generics)*, polymorfizmus, abstrakcia, a iné princípy objektovo orientovaného programovania v Jave. Okrem toho aj rekurzívne rozvetvenie volaní funkcií a spätná agregácia výsledkov.

Import zdrojového kódu je založený hlavne na zdedených a odvodených triedach modulu ANTLR (Parser, Lexer a Visitor), v ktorých je implementovaná kontrola a rozpoznávanie zdrojového kódu ako aj generovanie a skladanie objektov vývojového diagramu.

Export je založený na regulárnych výrazoch a aplikačnej logike s použitím triedy *FlowChart*, ktorá je konkrétnou implementáciou dátovej štruktúry *Diagram*.

Implementácia je ešte stále v procese. Pri prekladači Python sa ešte stále vyskytujú chyby. Po úspešnom dokončení budú tieto zmeny pridané do základného projektu PS Diagram a bude dostupný všetkým jeho používateľom, približne 5000 používateľov ročne. Vetva programu s prekladačom jazyka Java je dokončená a prebieha testovanie a zlučovanie s oficiálnou verziou programu PS Diagram.

PS Diagram tak bude ponúkať svojim používateľom všetky svoje funkcie, popísané vyššie, nie iba v programovacom jazyku Pascal ale v troch jazykoch, a to Pascal, Java aj Python. To prináša aj úplne novú možnosť vložiť algoritmus vytvorený v jednom z týchto jazykov, upraviť ho, skontrolovať a preložiť do ľubovoľného jazyka z výberu.

Ako možné rozšírenie aplikácie prichádzajú do úvahy ďalšie varianty programovacích jazykov. Zaujímavá by mohla byť myšlienka vytvárania viacerých inštancií Diagramu s možnosťou hierarchického usporiadania a vzájomnej komunikácie, formou volania a odpovede. Týmto spôsobom by PS Diagram mohol byť schopný prekladať nie len telo jednej metódy, ale viacero prepojených metód.

Zoznam použitej literatúry

[1] Robert Sedqewick, Kevin Wayne, Algorithms 4th edition https://algs4.cs.princeton.edu/home/ dostupné online 07.03.2021

[2] Skalka Ján, Cápay Martin, Lovászová Gabriela, Mesárošová Miroslava, Palmárová Viera; UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE; Algoritmizácia a úvod do programovania; 2007; ISBN 978-80-8094-217-5

https://encyklopediapoznania.sk/data/eknihy/informatika/algoritmizacia_a_uvod_do_programovania.pdf dostupné online 24.04.2021

[3] W3Consorcium dátové štruktúry https://www.w3schools.in/data-structures-tutorial/intro/ dostupné online 07.03.2021

[4] Michal Forišek, Monika Steinová, Explaining Algorithms Using Metaphors, Springer-Verlag GmbH, 2013, ISBN: 144715018X,

[5] Oficiálna dokumentácia programovacieho jazyka Java https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html dostupné online 07.03.2021

[6] Oficiálna webová stránka PS Diagram http://www.psdiagram.cz/prirucka dostupné online 07.03.2021

[7] Terence Parr, The Definitive ANTLR 4 Reference, Pragmatic Bookshelf, 2013, ISBN: 9781934356999

[8] Oficiálna dokumentácia ANTLR 4 - https://www.antlr.org/about.html dostupné online 07.03.2021

[9] zdrojový kód PS Diagram - https://github.com/PoloShock/psdiagram dostupné online 07.03.2021

[10] zdrojový kód ANTLR 4 - https://github.com/antlr/antlr4 dostupné online 07.03.2021

[11] Java 8 gramatický súbor - https://github.com/antlr/codebuff/blob/master/grammars/org/antlr/codebuff/Java8.g4 dostupné online 07.03.2021

[12] Sarah K. White, CIO, 9 legacy programming skills still in demand- https://www.cio.com/article/3243575/9-legacy-programming-skills-still-in-demand.html dostupné online 07.03.2021

[13] Brian Eastwood, Northeastern University, The most popular programming languages- https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/ dostupné online 07.03.2021

[14] Oficiálna dokumentácia programovacieho jazyka Python - https://docs.python.org/3/ dostupné online 07.03.2021

[15] Porovnanie programovacích paradigiem - https://en.wikipedia.org/wiki/Comparison_of_programming_paradigms dostupné online 31.03.2021

[16] Dr. Raaid Alubady – University of Babylon – College of Information Technology, publikácia s názvom Programming Languages: Classification, Execution Model and Errors http://www.uobabylon.edu.iq/eprints/publication_11_23917_6270.pdf

dostupné online 31.03.2021

[17] Ronald Mak; Writing Compilers and Interpreters: A Software Engineering Approach, Willey Publishing Inc. , 2009, ISBN: 978-0-470-17707-5

[18] Javatpoint, Classification of programming languages, https://www.javatpoint.com/classification-of-programming-languages

dostupné online 31.03.2021

[19] Jiří Jelínek, Ing. CSc., Katedra informatiky a přírodních věd, Ústav technicko-technologický, Vysoká škola technická a ekonomická v Českých Budějovicích PODPORA VÝUKY ALGORITMIZACE IT NÁSTROJI https://tvv-journal.upol.cz/pdfs/tvv/2016/01/17.pdf?fbclid=IwAR2jnOjloZB2z9TLcZa7Jwyd9xhQGz79ojnBe3h0OpmrsZ-9mdy-CPNLAWg

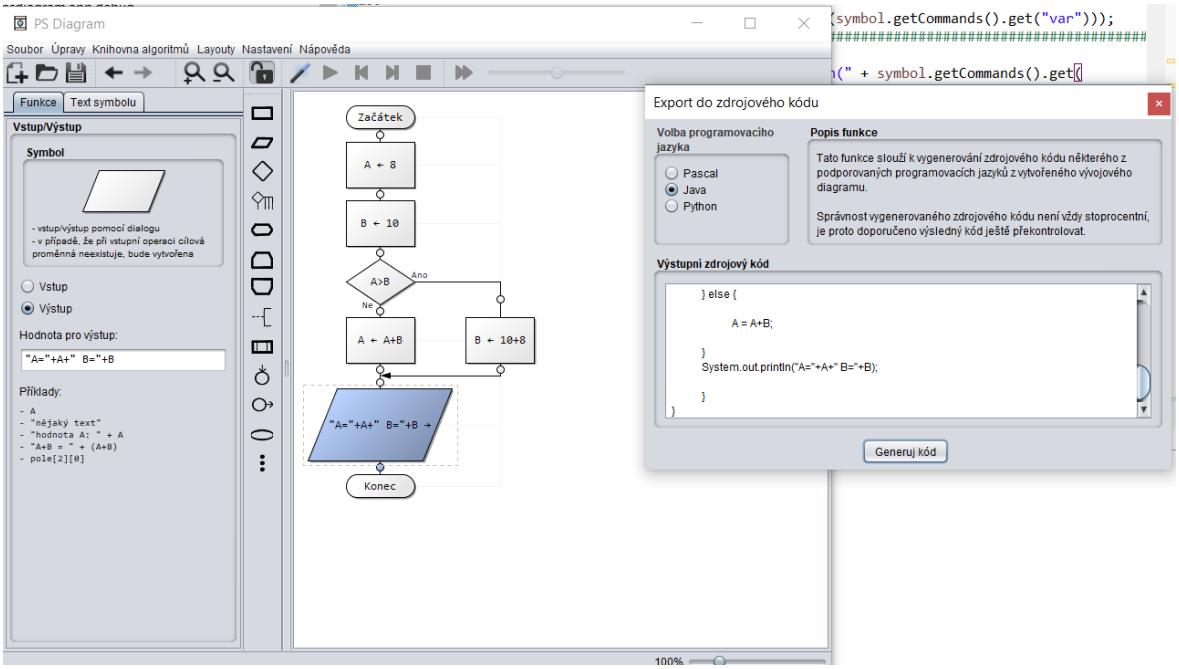
Dostupné online 06.04.2021

[20] Štatistiky získané z oficiálnej stránky PS Diagram, vlastné spracovanie

[21] Michael Schneider, Článok denníka INC, it’s harder to get into Google than Harvard https://www.inc.com/michael-schneider/its-harder-to-get-into-google-than-harvard.html navštívené 07.03.2021

[22] Grady Booch, James Rumbaugh, Ivar Jacobson; Unified Modeling Language User Guide, The, 2nd Edition; 2005, ISBN-13: 978-0-321-26797-9

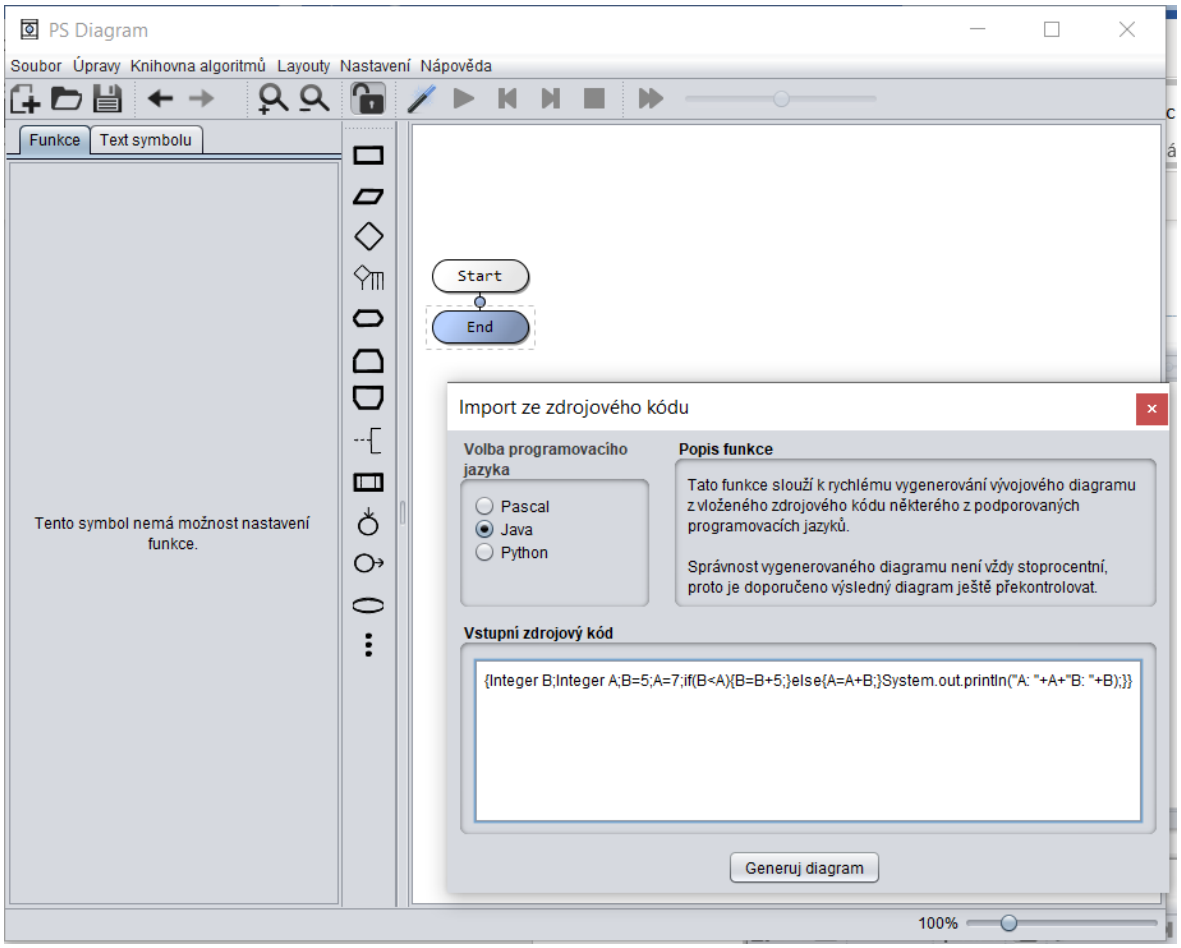
Prílohy



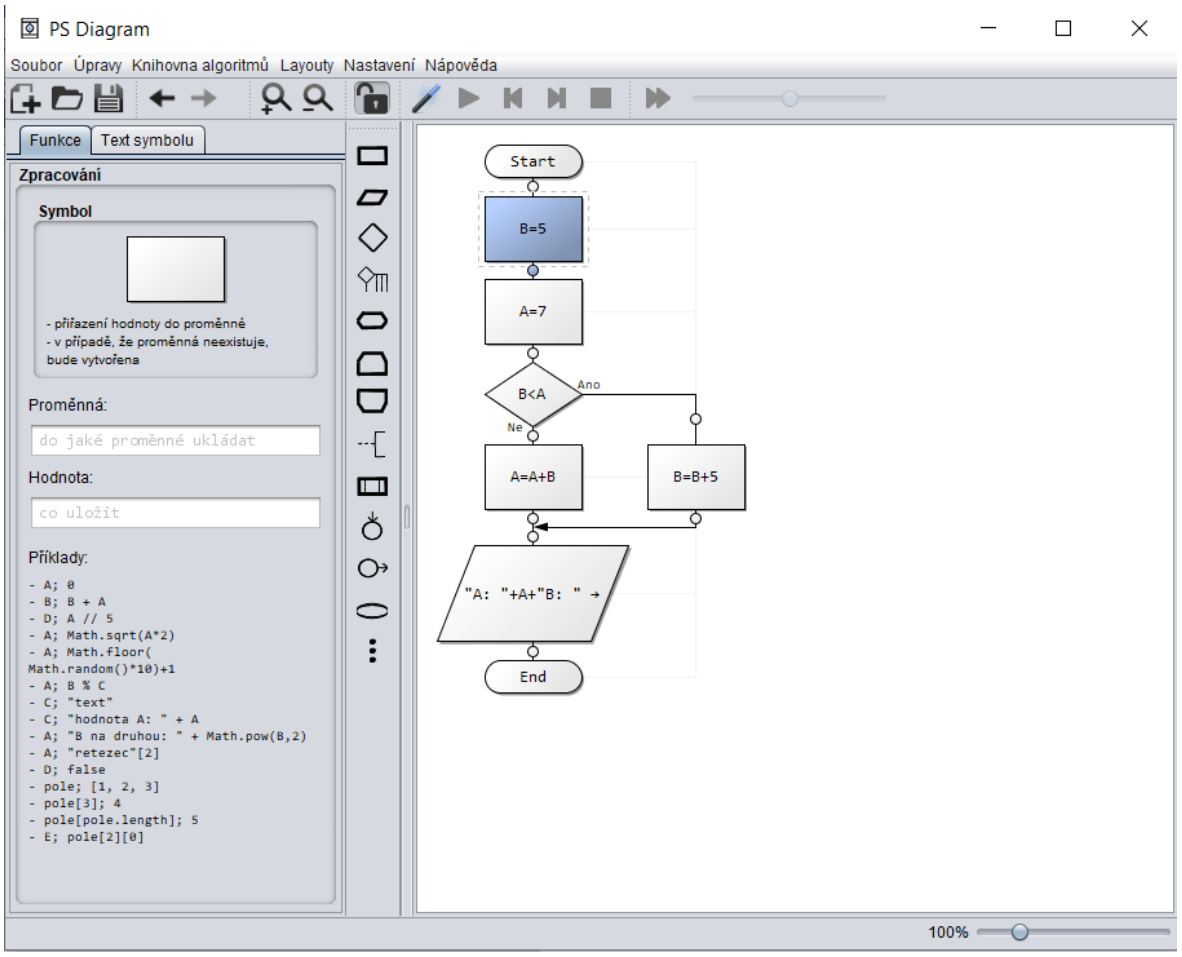
Príloha č.1 **Export zdrojového kódu Java** [vlastné spracovanie]

```
{
        Integer A;
        Integer B;
        A = 8;
        B = 10;
        if (A>B) {
                        B = 10+8;
        } else {
                        A = A+B;
        }
        System.out.println("A="+A+" B="+B);
}
```

Príloha č.2 **Exportovaný zdrojový kód Java** [vlastné spracovanie]



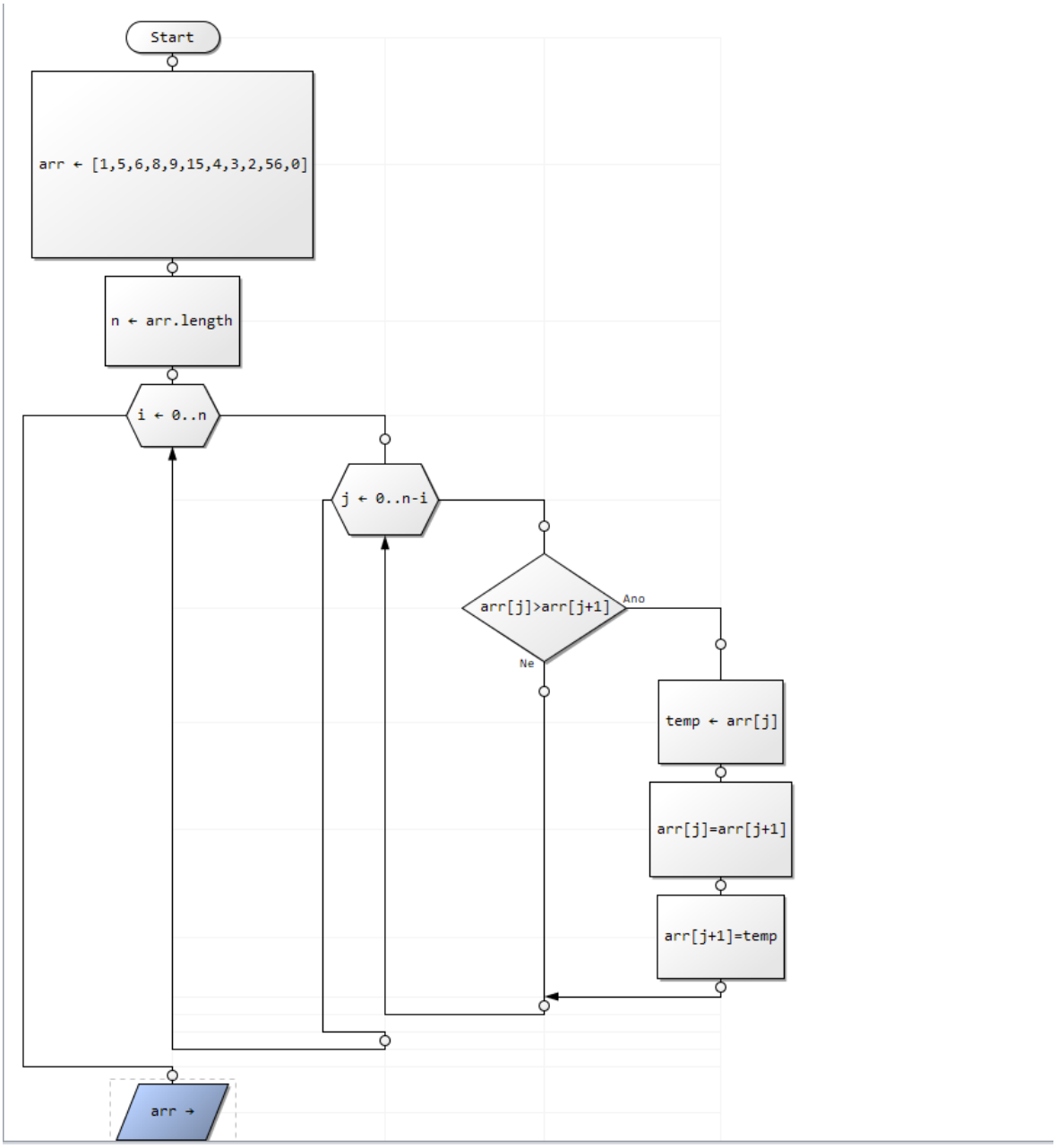
Príloha č.3 **Import zdrojového kódu Java** [vlastné spracovanie]



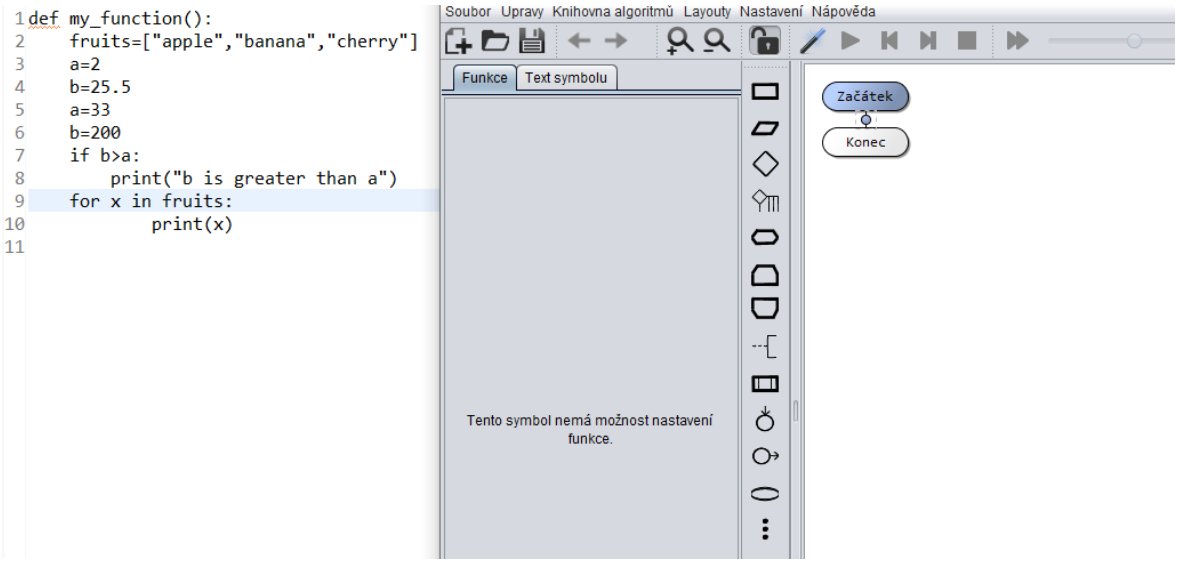
Príloha č.4 **Importovaný zdrojový kód Java** [vlastné spracovanie]

```
{
        int [] arr = {1,5,6,8,9,15,4,3,2,56,0};
        int n = arr.length;
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= n-i; j++) {
       if (arr[j] > arr[j+1]) {
          int temp = arr[j];
          arr[j] = arr[j+1];
          arr[j+1] = temp;
       }

        }
```

Príloha č.5 **Implementácia algoritmu Bubble Sort v Jave** [vlastné spracovanie]



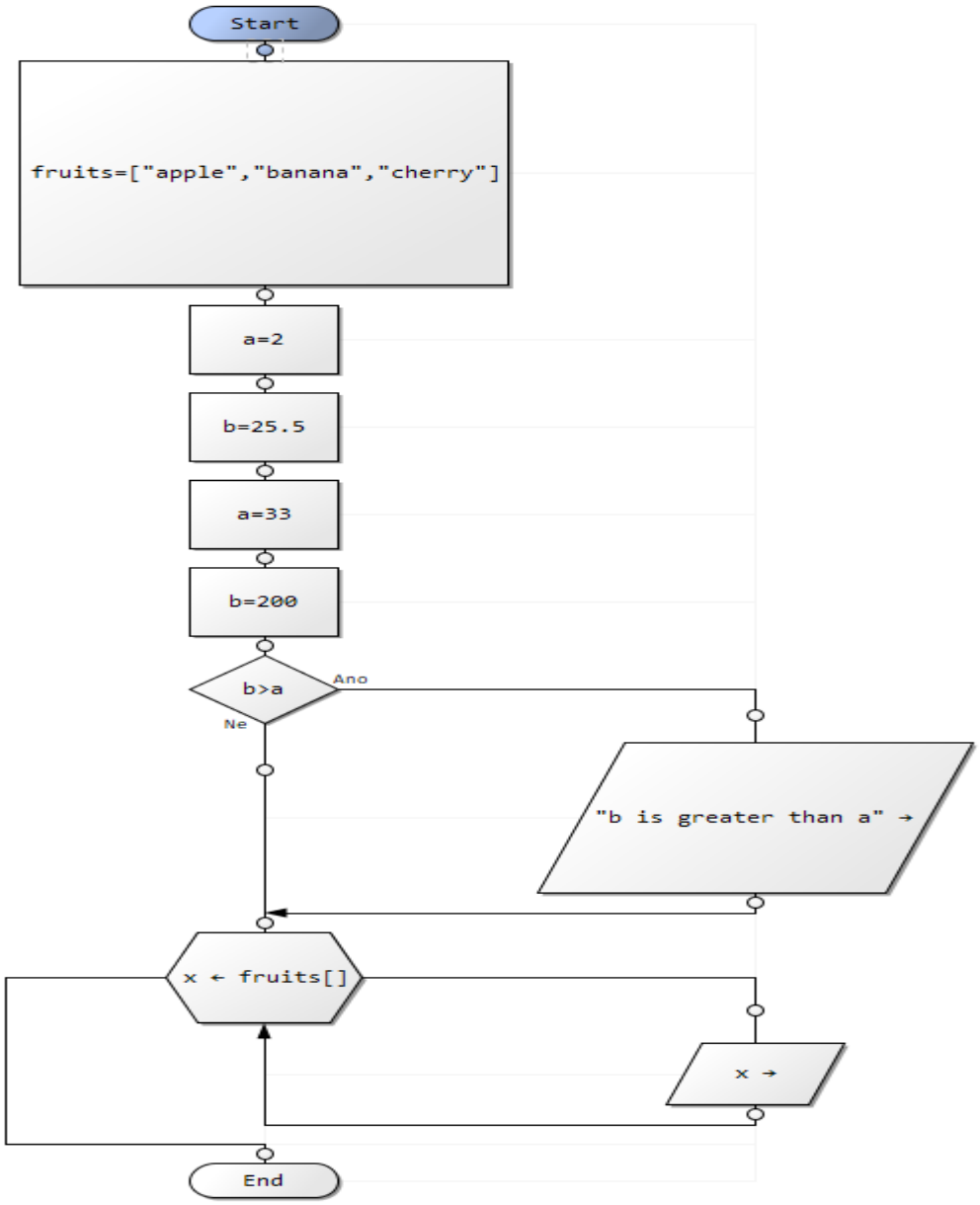
Príloha č.6 **Importovaný algoritmus Bubble Sort** [vlastné spracovanie]



Príloha č.7 **Import zdrojového kódu Python** [vlastné spracovanie]

```
def my_function():
        fruits=["apple","banana","cherry"]
        a=2
        b=25.5
        a=33
        b=200
        if b>a:
            print("b is greater than a")
        for x in fruits:
            print(x)
```

Príloha č.8 **Zdrojový kód Python** [vlastné spracovanie]



Príloha č.9 **Importovaný zdrojový kód Python** [vlastné spracovanie]

https://github.com/Jozef63/psdiagram/tree/Java_and_Python_parser_implementation

Príloha č.10 **Zdrojový kód implementovaný pri realizácií práce** [vlastné spracovanie]